Sistema d'entrenament per eolymp

Memòria del projecte

Victor Berral Navarro

Director:	Pau Fonseca i Casas
Departament:	Estadística i Investigació Operativa (EIO)
Titulació:	Enginyeria Técnica en Informàtica de Sistemes
Centre:	Facultat d'Informàtica de Barcelona (FIB)
Universitat:	Universitat Politècnica de Catalunya (UPC)
Data:	21/06/2013

DADES DEL PROJECTE

Títol del projecte:	Sistema d'entrenament per eolymp
Nom de l'estudiant:	Victor Berral Navarro
Titulació:	Enginyeria Tècnica en Informàtica de Sistemes
Crèdits:	22.5
Modalitat:	A
Director / Ponent:	Pau Fonseca i Casas
Departament:	Estadística i Investigació Operativa

MEMBRES DEL TRIBUNAL

Director:	Pau Fonseca i Casas
President:	Jose Casanovas Garcia
Vocal:	Maria Teresa Abad Soriano

QUALIFICACIÓ

Qualificació numèrica:

Qualificació descriptiva:

Data: 21/06/2013

Índex

1 Introducció

1.1 El projecte eolymp

Aquest projecte forma part del projecte de xarxa social eolymp, una xarxa oberta d'esportistes creada per investigadors de la Universitat Politècnica de Catalunya amb el suport de la iniciativa Microsoft® Pre-Incubation Program i ChampionChip.cat, l'empresa líder en cronometratge de curses a Catalunya.

Està destinada als corredors amb xip groc per tal que puguin consultar el seu historial de curses, compartir les seves marques amb altres corredors, alhora que veuen també els seus resultats i tenen la possibilitat d'inscriure's fàcilment a les curses de la Lliga Championchip des del mateix perfil.

Figura 1.1 Logo d'eolymp/ChampionChip

El web d'eolymp[1], a grans trets, compta amb una zona social on podem interaccionar amb d'altres usuaris, una zona de competició on veiem les curses en les que estem inscrits i un calendari assenyalant els propers esdeveniments esportius disponibles.

Figura 1.2 Captura del web eolymp.com

[1] Eolymp: http://www.eolymp.com

Tot i disposar d'un web bastant complet, hem de pensar que els usuaris, si segueix la tendència actual, accediran cada vegada més des d'un smartphone. Per tant, és lògic pensar que si disposen d'una aplicació mòbil que els ofereix un valor afegit es decantaran per aquesta.

Això ha comportat un creixement exponencial de les aplicacions específiques per a dispositius mòbils on, sobretot en aplicacions que bàsicament accedeixen a serveis i dades des d'internet i ens permeten operar-hi, es té molt en compte que la pantalla on es visualitza és molt més reduïda que la d'un ordinador, i per tant, la presentació de les dades ha d'estar focalitzada i la interacció amb l'aplicació serà amb els dits enlloc del punter del ratolí i un teclat.

Els dispositius mòbils als que ens referim, deixant de banda les populars tabletes, són els telèfons intel·ligents, més coneguts com a smartphones.

1.2 Smartphones i Sistemes Operatius

La tendència dels dispositius mòbils en general i smartphones en particular ha augmentat des de fa uns anys a un ritme imparable.

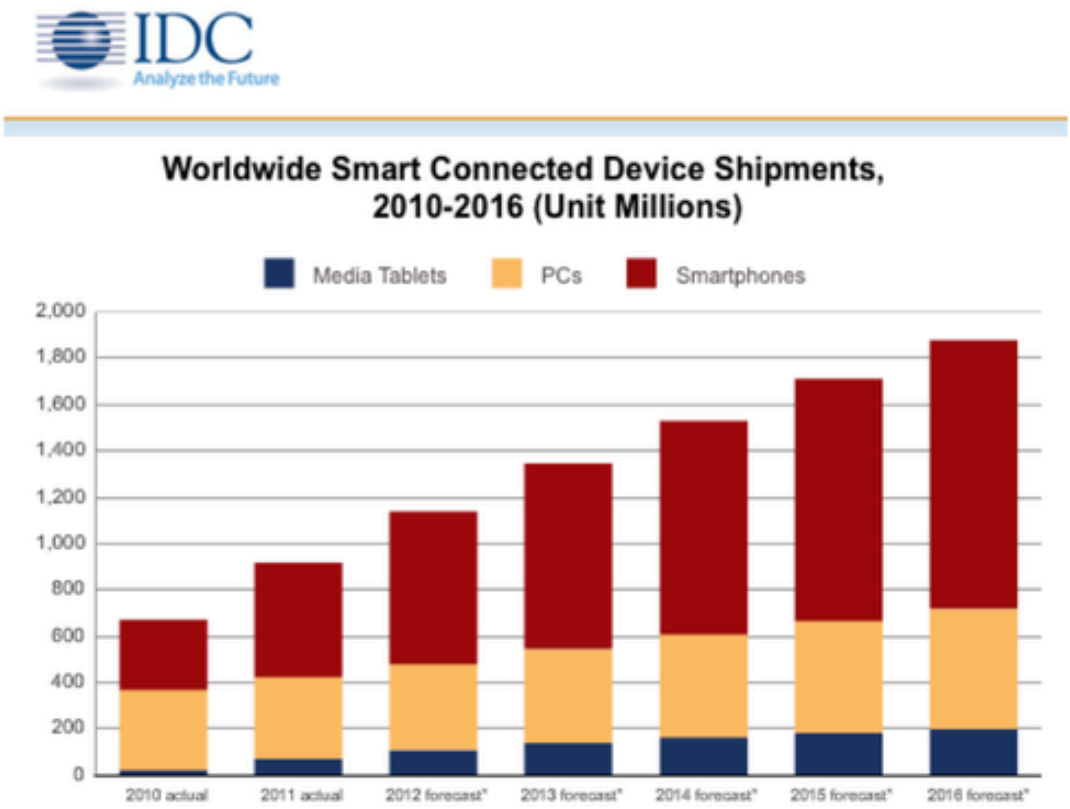


Figura 1.3 Smartphones connectats

Aquesta evolució ens indica que els usuaris cada vegada volen tenir més al seu abast, en tot lloc i moment les mateixes possibilitats que els hi dóna un ordinador fixe connectat a internet. És per això que el creixement del nombre d'aplicacions mòbils anirà augmentant, juntament amb el dels dispositius mòbils.

Un dels principals problemes als que s'enfronten els programadors d'aquestes aplicacions és que hi ha una gran varietat de sistemes operatius i per a cada plataforma existeix el seu propi llenguatge de programació.

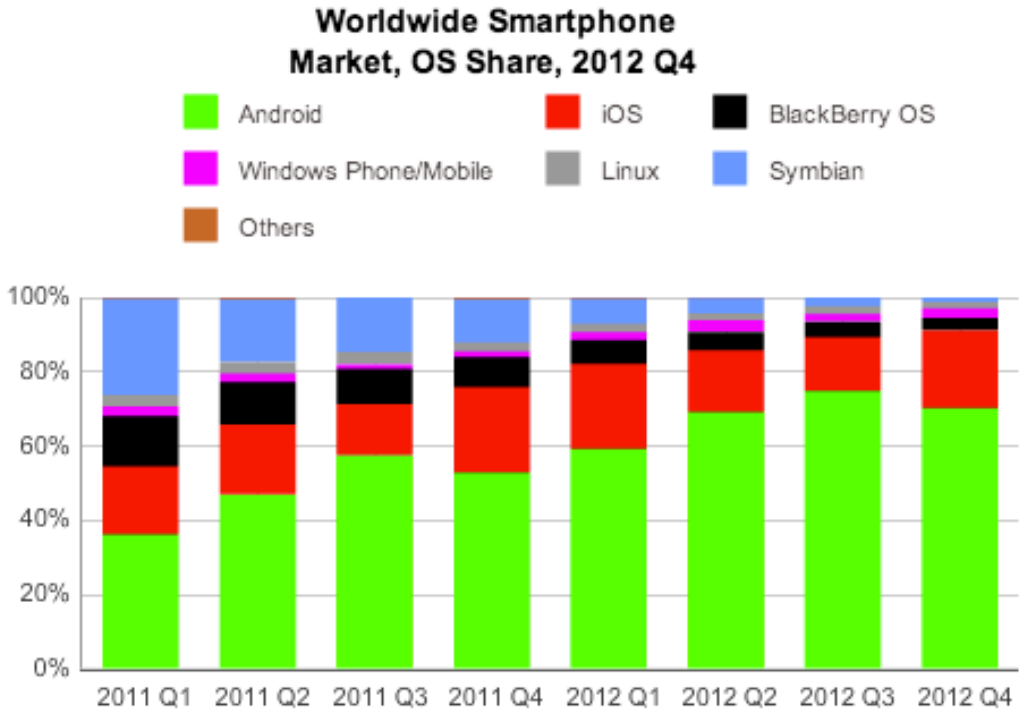


Figura 1.4 Sistemes operatius mòbils

En l'actualitat és Android el que disposa d'una major quota de mercat però els tres principals a tenir en compte, a banda d'Android, són iOS i, en una posició més discreta, Windows Phone. Com dèiem abans, cadascuna d'aquestes plataformes té el seu propi llenguatge de programació: per Android el llenguatge emprat es Java, per a Windows Phone és C# i per a iOS és Objective-C.

Tot i la semblança entre C# i Java, si volguéssim crear una aplicació per a les dues plataformes l'hauríem de programar en cadascuna d'elles. Un cas a part és Objective-C, que tot i ser també un llenguatge orientat a objectes, la

metodologia a seguir és bastant diferent dels altres dos i per tant, la seva corba d'aprenentatge és bastant elevada.

Ens trobem llavors amb que, si un programador vol crear una aplicació i oferir-la en els principals sistemes operatius mòbils, en realitat haurà de fer la mateixa feina per triplicat i, des del moment en que la publiqui, el manteniment d'aquesta també haurà de ser triple.

1.3 Desenvolupament unificat

Per tal d'estalviar-se el problema dels diferents llenguatges, durant els darrers anys han anat apareixent al mercat diferents opcions que ens permeten amb un sol llenguatge de programació i desenvolupant una única vegada l'aplicació, tenir-la disponible per a vàries plataformes.

Les principals alternatives són:

PhoneGap

Figura 1.5 Logo PhoneGap

Titanium

Figura 1.6 Logo Titanium

Rhomobile

Figura 1.7 Logo Rhomobile

Mono

Figura 1.8 Logo Mono

PhoneGap, Titanium i Rhomobile, tot i tenir les seves diferències, tenen en comú que són un conjunt d'eines que permeten, mitjançant tecnologies web com HTML, JavaScript i CSS, crear aplicacions del tipus web que poden córrer sobre la majoria de dispositius, independentment de la plataforma. Aquest, sens dubte, és un gran avantatge ja que aporta molta facilitat i flexibilitat al programador, però per contra les aplicacions resultants, al no ser natives, resulten poc eficients des del punt de vista del rendiment. També limita bastant si volem fer ús de llistes, conjunts de dades, connexions amb serveis web, etc.

L'altra alternativa és el projecte Mono, de l'empresa Xamarin. Aquest ens permet programar amb el llenguatge estàndard C#, que és un llenguatge molt madur i amb molta capacitat i que permet fer servir les llibreries .NET. Els avantatges d'aquest sistema són bàsicament dos: no disposarem de cap tipus de restricció de software i l'aplicació resultant és en codi natiu propi de la plataforma, per tant el rendiment d'aquesta serà òptim.

2 Objectius i motivacions

2.1 Objectius del PFC

L'objectiu d'aquest projecte és desenvolupar la base d'una aplicació mòbil que serveixi de client per a la xarxa eolymp. Ha de fer ús de la tecnologia Mono i el llenguatge C#, i compartir la major part possible de codi entre les plataformes Windows Phone, Android, iPhone i l'aplicació d'escriptori.

Des de l'inici ja es disposa de l'aplicació per a Windows Phone començada, i l'aplicació d'escriptori pràcticament acabada. El que es vol és extreure la major part possible de codi, unificar-lo i poder reutilitzar-lo en l'aplicació per a iOS i que totes quedin integrades dins d'un mateix projecte general de Visual Studio fent servir el repositori de codi ja existent.

Degut al temps limitat del que es disposa s'ha decidit deixar de banda l'aplicació per Android, tot i que tenint en compte el seu desenvolupament al futur, es vol deixar el codi el més preparat possible per facilitar la seva posterior addició a la solució final.

2.2 Motivació personal

La meva motivació personal en aquest projecte ha estat la d'intentar introduir-me en el món de la programació per a dispositius mòbils. Actualment treballo, entre d'altres coses, com a programador per a una empresa, i des de que va aparèixer el concepte d'aplicació mòbil ha estat una cosa que m'ha despertat molt d'interès, però per falta de temps i/o empenta mai havia aconseguit dedicar les hores necessàries per aprendre.

Aquest projecte per mi intentarà complir dos propòsits; d'una banda poder acabar la carrera, de la qual fa anys que només em faltava realitzar el projecte final per concloure, i d'altra banda adquirir els coneixements en la matèria de programació mòbil com a interès personal, i qui sap si algun dia, poder-me dedicar professionalment a aquesta branca de la informàtica, cosa que m'agradaria.

3 Eines de desenvolupament

3.1 Xcode i Simulador iOS

Per poder fer aplicacions per al sistema operatiu iOS (iPhone i iPad), Apple proporciona un kit de desenvolupament de software (SDK). Aquest kit conté bàsicament l'entorn de desenvolupament integrat (IDE) Xcode i un simulador per iOS.

Figura 3.1 Logo Xcode

Figura 3.2 Captura de pantalla del Simulador iOS amb Xcode de fons

Amb Xcode qualsevol programador disposa de tot el necessari per començar a desenvolupar aplicacions per iOS. El llenguatge que es fa servir per fer-ho és Objective-C, llenguatge orientat a objectes creat com a superconjunt de C, però com ja hem comentat anteriorment no serà l'emprat per realitzar aquest projecte i per tant no aprofundirem en el tema.

Xcode és gratuït i es va introduir l'any 2003 juntament amb la versió 10.3 de Mac OS X. Inclou una col·lecció de compiladors del projecte GNU i pot compilar codi C, C++, Objective-C, Objective-C++, Java i AppleScript mitjançant una àmplia gama de models de programació, incloent Cocoa i Java entre d'altres. Ha anat evolucionant fins la versió actual 4.6.2, que no és compatible amb versions anteriors a Mac OS X 10.7, més coneguda com a Lion.

Podríem separar Xcode en dos components ben diferenciats: Xcode en sí per a la compilació i depuració del codi, i Interface Builder per al disseny de les interfícies gràfiques d'usuari. Aquest últim és el que farem servir al projecte.

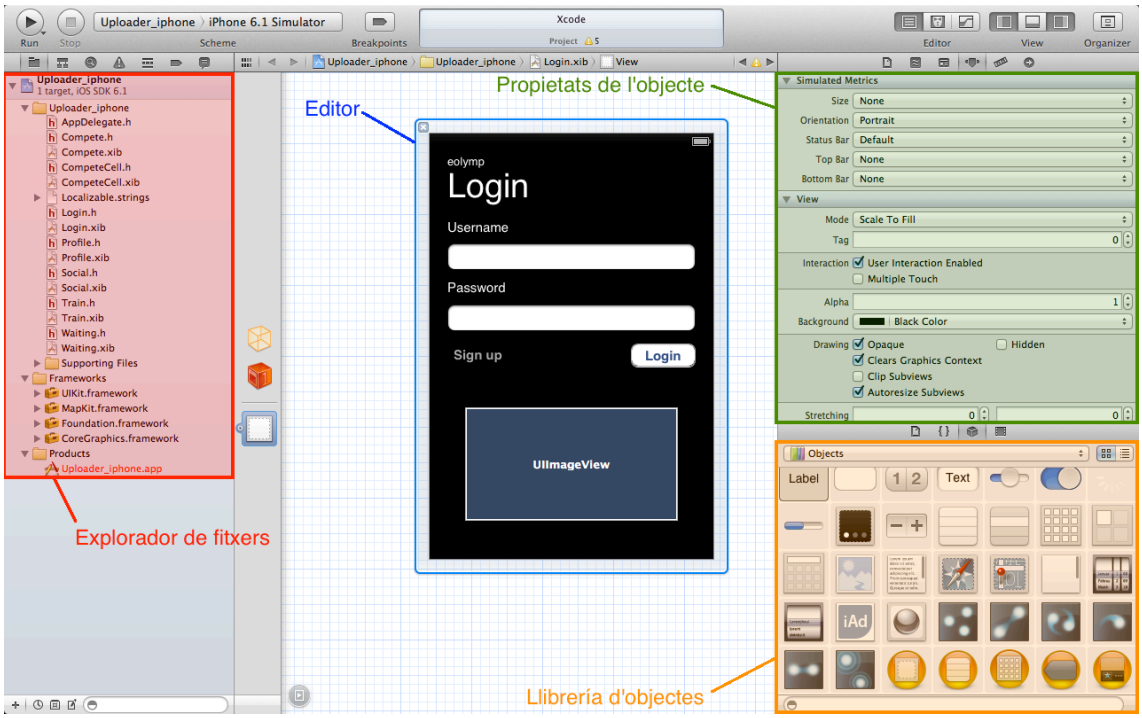


Figura 3.3 Interface Builder

Com s'aprecia a la Figura 3.3 la distribució de l'Interface Builder, a grans trets, es divideix en un explorador de fitxers, l'editor de la interfície, les propietats de l'objecte que estem tractant en aquell moment i una llibreria d'objectes per afegir a la nostra interfície.

La metodologia a seguir és senzilla; primerament cal dissenyar la pantalla afegint tots els objectes que necessiti la nostra vista a l'editor, elements disponibles a la llibreria tals com botons, taules, contenidors d'imatges, etiquetes, etc. Un cop afegits a la pantalla i col·locats al seu lloc determinat podem personalitzar les propietats de cadascun d'ells a l'apartat de propietats tot seleccionant l'objecte a tractar a l'editor i buscant la propietat que volem modificar, com ara el color o la mida, i canviant-la allà.

Amb això obtindrem una vista dissenyada segons les nostres necessitats, com hauríem fet amb qualsevol IDE, però amb un problema i és que amb Xcode s'ha de realitzar un pas més per poder accedir als objectes des del codi del programa. És el que s'anomena crear els Outlets i les Actions.

Crear un Outlet significa que fem l'objecte accessible des del codi per tal de poder canviar les seves propietats en temps d'execució, com ara afegir text si es tracta d'una caixa de text. Quan creem l'Outlet ens obliga a posar-hi un nom i aquest serà el nom amb el que podrem accedir des del codi font.
Un Action funciona de la mateixa manera però el que ens permet és capturar l'acció sobre un objecte, com ara quan l'usuari ha clicat un botó, i reaccionar en conseqüència.

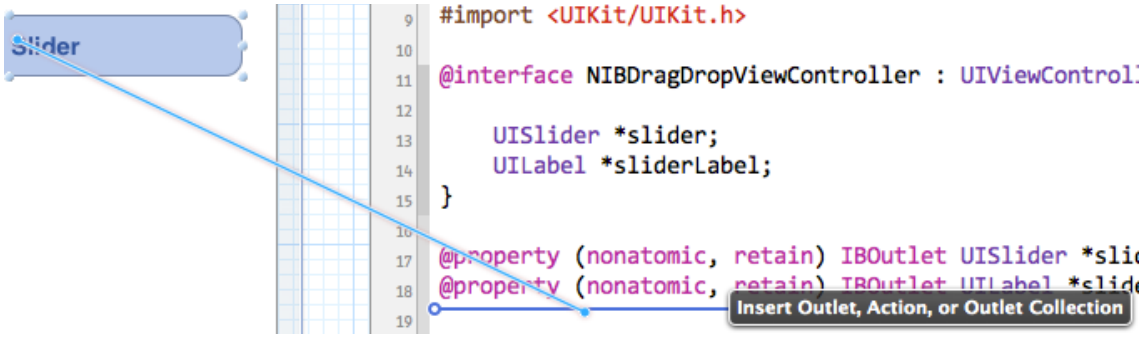


Figura 3.4 Detall 1 de creació d'un Outlet/Action

Per crear un Outlet o un Action haurem de mostrar la part de codi de la interfície, cosa que podem fer mitjançant el menú (View → Assistant editor → Show assistant editor), i un cop mostrada i mantenint polsada la tecla CTRL, arrastrem l'objecte dins de la part del codi tal i com es veu a la Figura 3.4.

A continuació ens apareixerà una petita finestra on haurem d'indicar si el que volem es un Outlet o un Action i el nom que li volem posar (Figura 3.5).

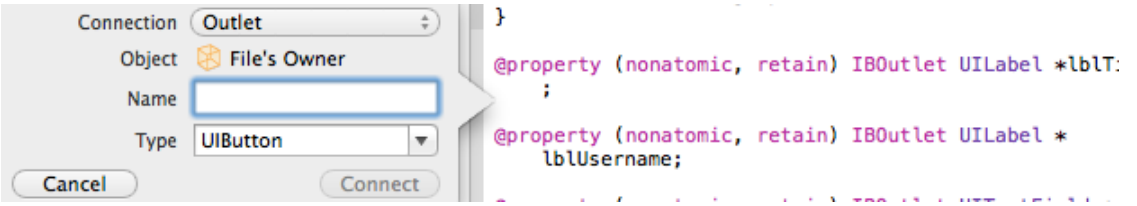


Figura 3.5 Detall 2 de creació d'un Outlet/Action

Un cop dissenyada la interfície i afegits els Outlets i els Actions necessaris, el resultat s'emmagatzema en un fitxer amb extensió .xib amb el que ja podem treballar al nostre IDE sense necessitat de fer servir Xcode ni Objective-C per programar la nostra aplicació, tot això gràcies a l'ús de les llibreries MonoTouch.

3.2 Xamarin i el projecte Mono

Tal i com presentàvem a la introducció d'aquest document, Xamarin[2] ens ofereix la possibilitat de desenvolupar aplicacions per a dispositius mòbils, ja sigui per a Windows Phone, Android o iPhone, fent ús del llenguatge C#, amb la possibilitat de compartir gran part del codi entre plataformes i amb l'avantatge que el resultat és una aplicació nativa de la plataforma de destí.


Figura 3.6 Logo Xamarin

El projecte Mono fou llençat per primera vegada l'any 2004. Es tracta d'un projecte de codi obert iniciat per Ximian i posteriorment impulsat per Novell (un cop que va adquirir Ximian) per a crear un conjunt d'eines lliures basades en GNU/Linux i compatibles amb .NET, un cop convertit aquest últim en un estàndard. A l'Abril de 2011 Novell havia estat adquirida per l'empresa

[2] Xamarin: http://www.xamarin.com

Attachmate i es va anunciar l'acomiadament de molts treballadors, incloent-hi els responsables del projecte Mono. Això va provocar, només un mes més tard, al maig de 2011, el naixement de l'empresa Xamarin, formada en part pels enginyers que van crear Mono i havien estat acomiadats, amb Miguel de Icaza al capdavant.
Xamarin es va comprometre a donar continuïtat i suport al projecte Mono, juntament amb una sèrie de productes específics per a dispositius mòbils, com ara Xamarin.iOS (anteriorment Mono Touch).

Però com funciona Mono? Per entendre el funcionament primer cal estar familiaritzat amb dos termes dels quals en fa ús: CLI i CLR.

El CLI o infraestructura de llenguatge comú (Common Language Infrstructure) és una especificació estandarditzada que descriu un entorn virtual per a l'execució d'aplicacions, amb la principal característica de permetre que aplicacions escrites en diferents llenguatges puguin després executar-se en múltiples plataformes sense la necessitat de reescriure o recompilar el seu codi font.

D'altra banda, el CLR o entorn en temps d'execució de llenguatge comú (Common Language Runtime) és un entorn d'execució per als codis dels programes que corren sobre la plataforma Microsoft .NET. És l'encarregat de compilar el codi intermediari en CLI al codi de màquina natiu mitjançant un compilador en temps d'execució.

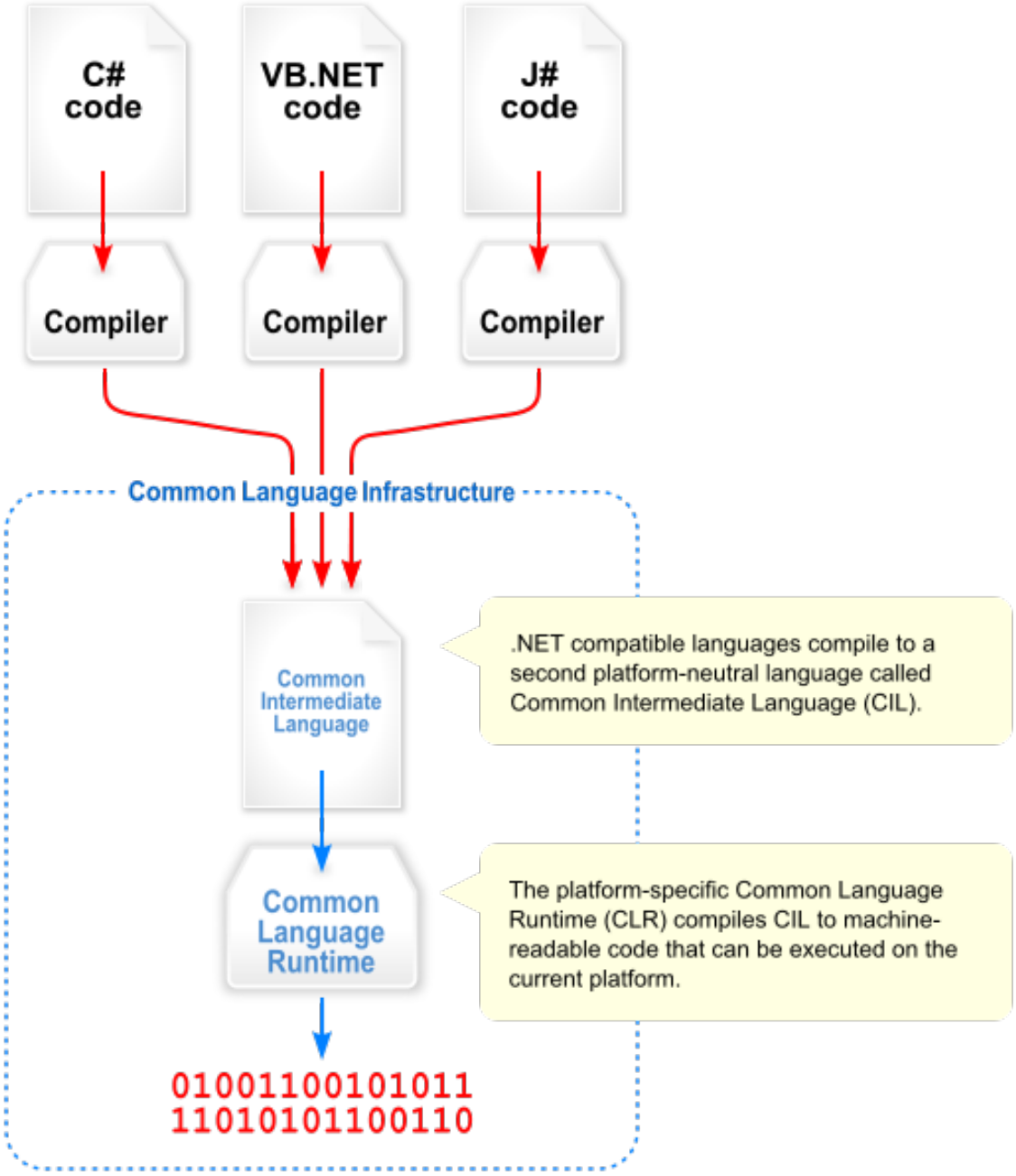


Figura 3.7 Esquema de funcionament de CLI i CLR

Observant l'esquema de funcionament de CLI i CLR (Figura 3.7) queda clar que aquesta tecnologia ens permet programar una aplicació amb C# i després el compilador s'encarrega de transformar-la en codi natiu. Això és exactament el que ens permeten fer les llibreries Xamarin.iOS i Xamarin.Android (per iOS i Android respectivament) i aprofitem aquí per desenvolupar la nostra aplicació.

Xamarin proporciona també un IDE gratuït per tal de poder desenvolupar en un entorn força complet el nostre programa, conegut amb el nom de Xamarin Studio (anteriorment Mono Develop).

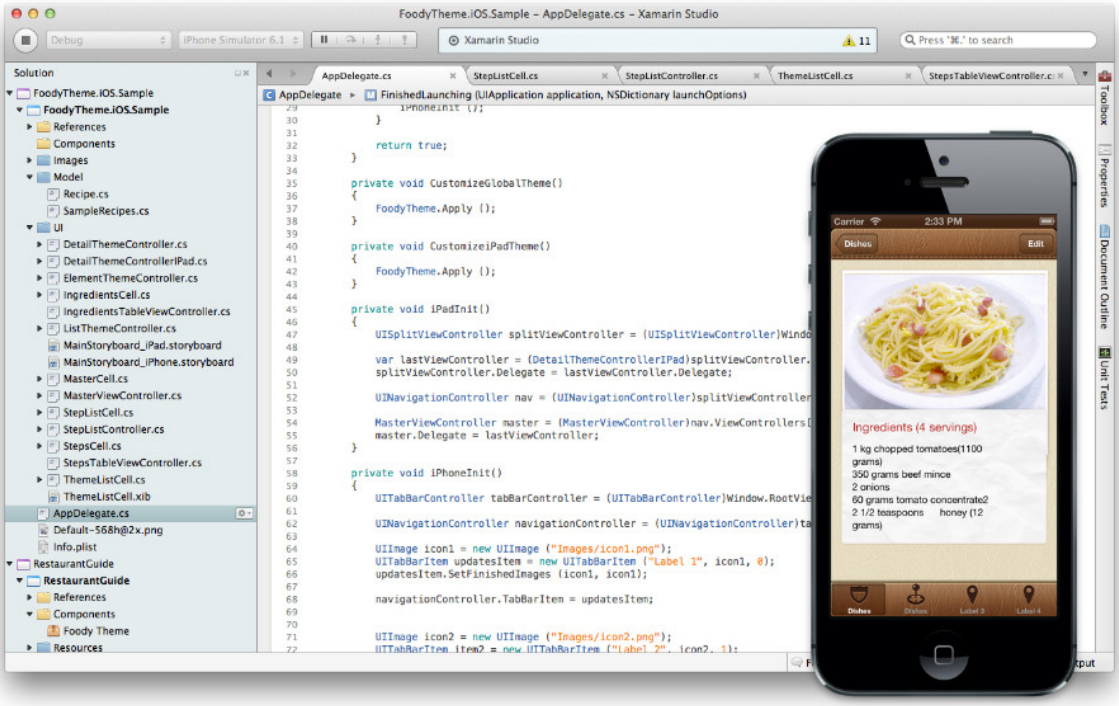


Figura 3.8 Captura de pantalla de Xamarin Studio

Aquesta aplicació s'ha començat desenvolupant amb aquest IDE per comoditat, ja que al principi només es podia executar el simulador d'iPhone des d'aquest IDE.

Amb la darrera versió de les llibreries Xamarin sí que es permet connectar tot plegat amb el Visual Studio i executar el simulador des d'allà.

3.3 C# i Objective-C

Tot i no ser una eina pròpiament dita, amb el llenguatge de programació C# obtenim alguns avantatges a l'hora de desenvolupar l'aplicació, sobretot comparat amb el llenguatge natiu d'iOS, l'Objective-C. Per aquesta raó és digne de menció en aquest document.

Per situar-nos cal comentar que C# és un llenguatge de programació orientat a objectes creat per Microsoft i estandaritzat més endavant, que forma part de la plataforma .NET. Com hem pogut observar en el capítol anterior, va ser dissenyat per a funcionar amb la infraestructura de llenguatge comú (CLI).

Objective-C, per la seva banda, també és un llenguatge de programació orientat a objectes però la seva estructura fa que sigui més difícil de programar-hi que amb el C# (podem veure un exemple a la Figura 3.9).

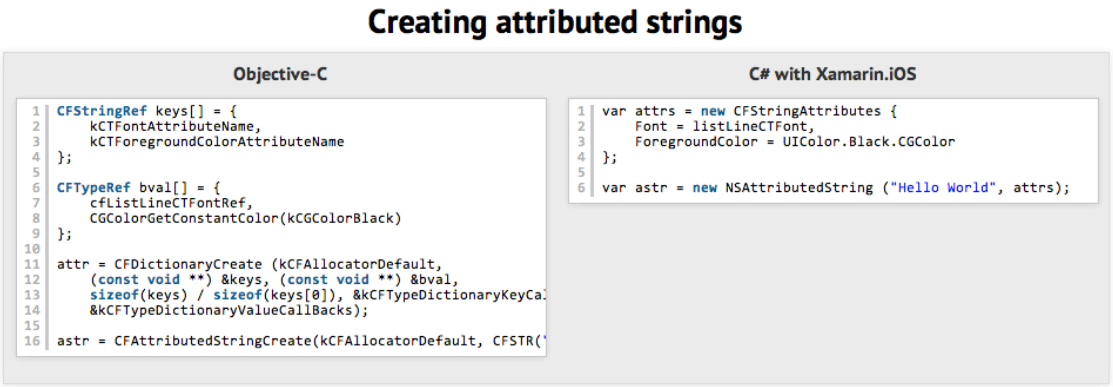


Figura 3.9 Exemple de diferències entre C# i Objective-C

També la seva semblança amb altres idiomes com Java, C o C++ fa reduir pràcticament a zero la seva corba d'aprenentatge, just al contrari que amb Objective-C.

Altres característiques que aprofitarem en el desenvolupament d'aquesta aplicació són les anomenades **directives de compilació condicional**. Aquestes ens permeten incloure o excloure de forma condicional parts del codi font, de manera que en un mateix fitxer podem crear tota una classe amb una funció determinada, com ara tot el necessari per fer el login i, en el cas que hi hagi petites diferències en la manera de fer-ho en iPhone, Windows Phone o Android, afegint aquestes etiquetes podem especificar quina part pertany a quin sistema per tal que el compilador només la tingui en compte si estem treballant amb aquest sistema.

```
349         public XmlRpcStruct SystemConnect()
350         {
351 #if DESKTOP
352             return m_drupal.SystemConnect();
353 #elif MONOTOUCH
354             var proxy=new DrupalProxy();
355 
356             m_result = proxy.BeginSystemConnect(asr =>
357             {
358                 m_UIApplication.BeginInvokeOnMainThread (delegate {
359                     try
360                     {
361                         m_drupalData = proxy.EndSystemConnect(asr);
362                         Console.WriteLine("CONNECT!");
363                         if (m_drupalData == null) m_errConnec = true;
364                     }
365                     catch (XmlRpcFaultException fex)
366                     {
367                         m_errConnec = true;
368                         funcions.mostrarMissatgeOK("[" + fex.FaultCode.ToString() + "] " + fex.FaultString);
369                     }
370                     catch (Exception ex)
371                     {
372                         m_errConnec = true;
373                         if (ex.Message == "") funcions.mostrarMissatgeOK("No connection");
374                         else funcions.mostrarMissatgeOK(ex.Message);
375                     }
376                 });
377             });
378             return m_drupalData;
379 #else
380             var proxy = new DrupalProxy();
381             m_result = proxy.BeginSystemConnect(asr =>
382             {
383                 m_Dispatcher.BeginInvoke(delegate()
384                 {
385                     try
```

Figura 3.10 Exemple de compilació condicional

A la Figura 3.10 podem observar com, si estem treballant en l'entorn per iPhone i tenim definida aquí l'etiqueta (també anomenat símbol) de compilació condicional "MONOTOUCH", el sistema només reconeixerà com a codi les parts assenyalades amb aquesta etiqueta, i per tant el que quedi fora no afecta al nostre codi en aquesta plataforma.

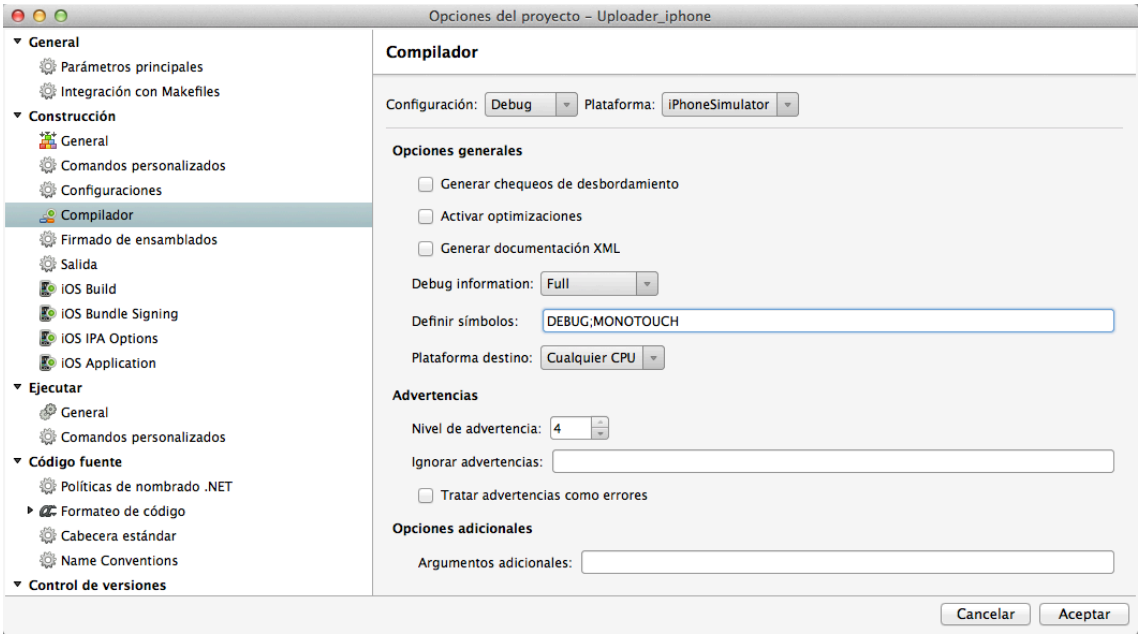


Figura 3.11 Definició d'etiquetes de compilació condicional

3.4 Drupal i XML-RPC

La nostra aplicació és un client que és connecta a un servidor web per rebre la informació i mostrar-la per pantalla, a més de permetre'ns interactuar-hi.
Per facilitar la tasca, el servidor web compta amb Drupal, un sistema gestor de continguts modular i molt configurable, de codi obert i gratuït.

Figura 3.12 Logo Drupal

El servidor real al que es connectarà l'aplicació està ja en funcionament, però per fer les proves pertinents crearem una rèplica local amb el mínim necessari a la màquina on es desenvolupa el projecte.

Aquestes són algunes de les característiques de les que farem ús:

- Permet estructurar la informació de forma dinàmica ja que la guarda en una base de dades MySQL.
- Ofereix gestió integrada d'usuaris i contrasenyes.
- Permet crear tipus de contingut personalitzats i treballar amb ells (nodes).
- És configurable mitjançant interfície web.
- Permet la comunicació mitjançant XML-RPC, gràcies al mòdul servidor de XML-RPC.

Arribats a aquest punt hem de donar èmfasi al XML-RPC, un protocol que permet fer crida de procediments remots (emmagatzemats al servidor web) a través d'internet fent servir el protocol HTTP i codificant la resposta en el llenguatge de marques XML. En concret en aquest projecte farem servir XML-RPC.NET, una llibreria que permet fer ús dels serveis XML-RPC en l'entorn .NET.

XML-RPC.NET representa els mètodes d'un servidor XML-RPC com si es tractessin de mètodes .NET. Per fer crides a un servidor XML-RPC cal fer servir una instància de la classe proxy. Veiem ara un exemple de crida a una funció d'un servidor XML-RPC dintre de codi .NET:

En primer lloc, cal crear una interfície que representi el mètode del servidor al que volem accedir mitjançant l'atribut *XmlRpcMethod* i el converteixi en un *XmlRpcProxy*:

```
using CookComputing.XmlRpc;

public struct SumAndDiffValue
{
     public int sum;
     public int difference;
}

[XmlRpcUrl("http://www.cookcomputing.com/sumAndDiff.rem")]
public interface ISumAndDiff : IXmlRpcProxy
{
     [XmlRpcMethod]
     SumAndDiffValue SumAndDifference(int x, int y);
}
```

Després, cal crear una instància d'una classe proxy creada dinàmicament:

```
ISumAndDiff proxy = XmlRpcProxyGen.Create<ISumAndDiff>();
```

Per últim, podem fer la crida al mètode:

```
SumAndDiffValue ret = proxy.SumAndDifference(2, 3);
```

D'aquesta manera hem aconseguit cridar al mètode *SumAndDifference* que està al servidor i rebre el resultat dins de la variable *ret*.

Una altra avantatge del XMLRPC, a banda de donar accés a mètodes del servidor, són els *XmlRpcStructs*. Aquests objectes proporcionen al programador molta facilitat a l'hora de crear una estructura de dades per poder intercanviar amb el servidor.

Per exemple, imaginem el cas que volem enviar al servidor informació sobre una nova activitat, com ara el nom de l'activitat, la durada d'aquesta i les persones que han participat. Si enviéssim aquesta informació continguda en un XmlRpcStruct tindria la següent forma:

```
<struct>
  <member>
    <name> nom_activitat </name>
    <value> <string> futbol </string> </value>
  </member>

  <member>
    <name> durada_activitat </name>
    <value> <i4> 90 </i4> </value>
  </member>

  <member>
    <name> persones_implicades </name>
    <value>
      <array>
        <data>
          <value> <string> Joan </string> </value>
          <value> <string> Laura </string> </value>
          <value> <string> Pere </string> </value>
        </data>
      </array>
    </value>
  </member>
</struct>
```

El que ens diu aquesta estructura és que enviarem el camp *nom_activitat* amb el valor textual *futbol*, per al camp *durada_activitat* enviarem el valor de tipus enter *90*, i per al camp *persones_implicades* enviarem una col·lecció (*array*) de valors textuals: *Joan, Laura* i *Pere*, que podria ser tan gran com vulguem.

Com es pot apreciar aquest sistema de codificació per l'enviament de dades ofereix moltes possibilitats, ja que al ser recursiu podem afegir tantes parelles membre-valor com es desitgi, i fins i tot com hem vist podem enviar col·leccions de valors. Els valors poden ser dels tipus estàndard: text, enter, decimal, booleà, etc.

El que acabem de veure és com els valors viatgen a través de la xarxa en llenguatge XML, però si volem fer-ho servir al nostre codi, gràcies als XmlRpcStructs no haurem de construir tota l'estructura nosaltres, sinó que podem crear un objecte d'aquest tipus i anar afegint els valors desitjats.

Per reproduir l'exemple anterior en C#, un XmlRpcStruct amb la mateixa forma es construiria de la següent manera:

```
XmlRpcStruct estructura = new XmlRpcStruct();
estructura.add ("nom_activitat", "futbol");
estructura.add ("durada_activitat", 90);
Object[] persones = {"Joan", "Laura", "Pere"};
estructura.add ("persones_implicades", persones};
```

3.5 Subversion

Per tal de mantenir l'estructura correcte del codi, poder compartir-ho entre les diferents aplicacions i mantenir un control sobre els canvis realitzats, en aquest projecte s'ha treballat amb un repositori *subversion*.

Subversion, anomenat també *svn* per l'eina emprada per línia de comandes, és un sistema de control de versions lliure i de codi font obert. El seu funcionament és igual al d'un servidor de fitxers estàndard amb la diferència que recorda tots els canvis fets als fitxers i directoris. Aquesta utilitat permet recuperar versions anteriors dels fitxers, així com revisar l'historial de canvis realitzats en ells.

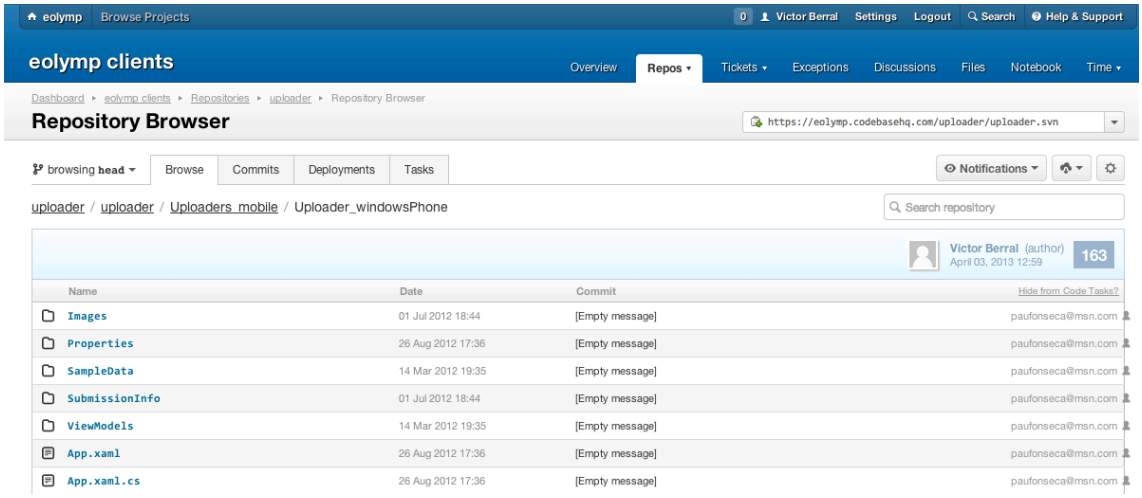


Figura 3.13 Detall del repositori web

Totes aquestes característiques fan que l'eina sigui una molt adequada forma de mantenir codi font on col·laboren diferents persones i a través de la xarxa.

Tot i poder accedir al repositori svn a través de la web, tal i com es veu a la Figura 3.13, i mitjançant la línia de comandes, per realitzar aquest projecte s'ha fet servir VISUALSVN, un plugin per a Visual Studio que ens permet accedir al repositori de manera molt fàcil i visual.

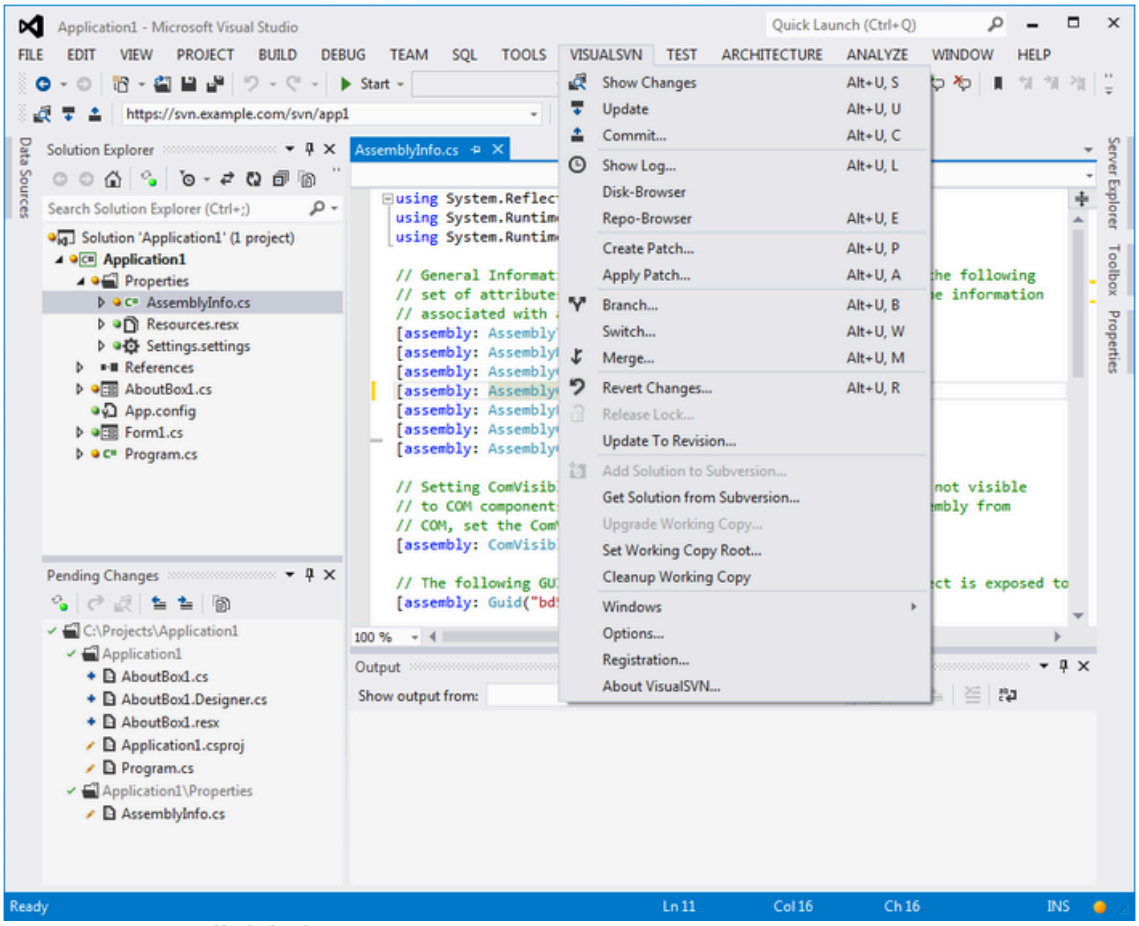


Figura 3.14 Detall del plugin VISUALSVN

Si observem la Figura 3.14 veiem el menú desplegable d'aquest plugin per a Visual Studio. La manera de procedir seria anar escrivint i/o modificant el codi font i quan ja està testejat i estem segurs que funciona correctament faríem servir la comanda **Commit** que es veu al menú contextual de la captura. Aquesta agafa el fitxer en qüestió i l'actualitza al servidor, afegint una marca conforme el codi que estem fent servir és la versió més actual.

D'altra banda, si el codi ha estat modificat des d'una altra aplicació i volem actualitzar aquest en la actual solució, hauríem de fer-ho mitjançant la opció **Update**. Aquesta descarrega del servidor la última versió del fitxer i substitueix l'actual.

4 Hardware utilitzat

Per desenvolupar aquesta aplicació s'ha fet servir un portàtil MacBook Pro amb pantalla de 13,3 polzades amb les següent especificacions:

- Processador Intel Core i5 a 2,5 GHz
- 16 GB de memòria RAM DDR3 a 1600 MHz
- 500 Gb de disc dur
- Tarja gràfica HD Graphics 4000 d'Intel
- Sistema operatiu OS X 10.8.3 (Mountain Lion)

Figura 4.1 MacBook Pro de 13

Per poder executar Visual Studio 2012 és necessari també comptar amb un ordinador amb el sistema operatiu Windows 8 però per comoditat s'ha fet servir el mateix MacBook Pro amb el Windows virtualitzat mitjançant l'eina Parallels Desktop 8 per a Mac.

Figura 4.2 Captura de pantalla de Windows rodant amb Parallels Desktop

D'aquesta manera disposem de tot el necessari per desenvolupar l'aplicació en un sol equip, cosa que facilita les tasques.

5 Disseny inicial de l'aplicació Windows Phone

5.1 Pantalles de Windows Phone

Des de l'inici d'aquest projecte ja es disposava d'una aplicació per a Windows Phone i, per tant, s'ha volgut seguir l'esquema ja emprat per desenvolupar la nova aplicació.

En obrir l'aplicació apareix una pantalla on l'usuari es pot connectar amb el seu nom d'usuari i contrasenya:

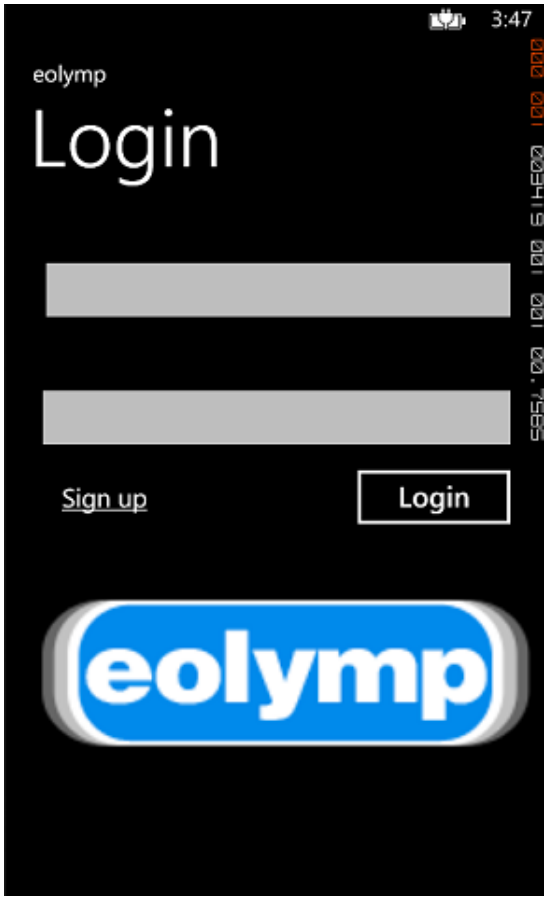


Figura 5.1 Pantalla de login

Figura 5.2 Pantalla d'espera

Si no tenim creat un usuari a eolymp també podem crear-ne un de nou mitjançant el botó Sign up, que ens redirigeix al web d'eolymp per tal de poder donar-nos d'alta.

Un cop l'usuari es valida correctament i mentre esperem que l'aplicació rebi les dades del servidor apareix una pantalla d'espera.

Passat un interval de temps variable, que depèn de la velocitat de la connexió i del volum de dades que s'han de descarregar, accedim a la pantalla principal de l'aplicació. Aquesta es divideix en tres grans apartats: zona de competicions, zona social i informació del nostre perfil.

Des de la pantalla de competicions podem veure una llista amb totes les competicions que hi ha disponibles en aquest moment. Veiem el títol de la competició, la data i una icona que ens indica qui ha estat el creador de l'esdeveniment.

Si polsem en una d'elles, el sistema ens dirigeix al navegador web del nostre telèfon obrint la pàgina de la cursa triada i mostrant-nos tots els detalls.

Figura 5.3 Pantalla competicions

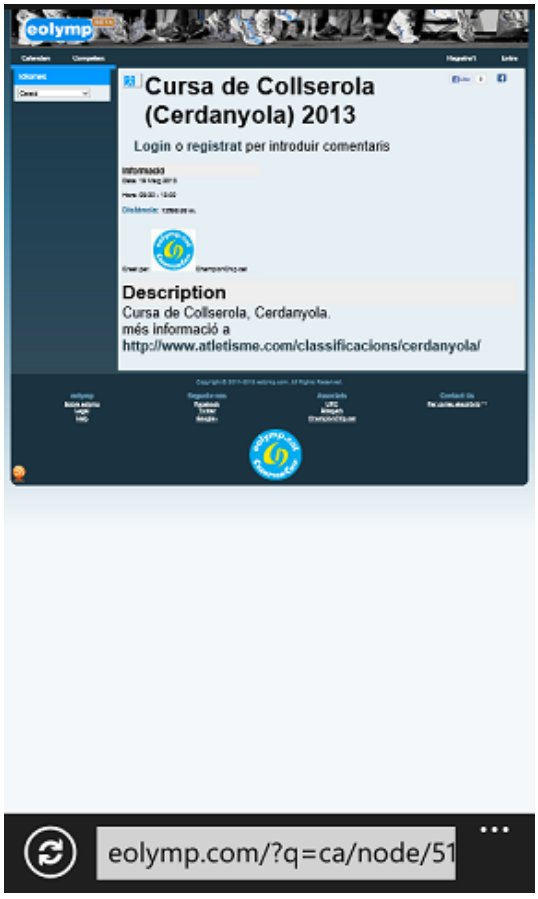


Figura 5.4 Detall competició

Si tanquem el navegador i tornem a l'aplicació podem passar a la pantalla següent, que és la pantalla social. Per alternar entre aquestes tres pantalles es fa servir el control panorama de Windows Phone, que ve a ser com si les tres pantalles fossin una de sola situades una a continuació de l'altre i ens permet desplaçar-nos lliscant el dit a esquerra o dreta per mostrar la següent pantalla.

Visualment, la pantalla social és igual que l'anterior de competicions. En aquesta veiem una llista dels comentaris que han fet els usuaris, tenint com a títol el nom de la competició sobre la que tracta el comentari, com a subtítol les primeres paraules del comentari i com a imatge la foto de l'usuari que ha fet el comentari.

Si polsem sobre qualsevol d'ells, de la mateixa manera que abans, l'aplicació ens torna a redirigir al navegador i ens mostra la pàgina web amb el comentari.

Figura 5.5 Pantalla social

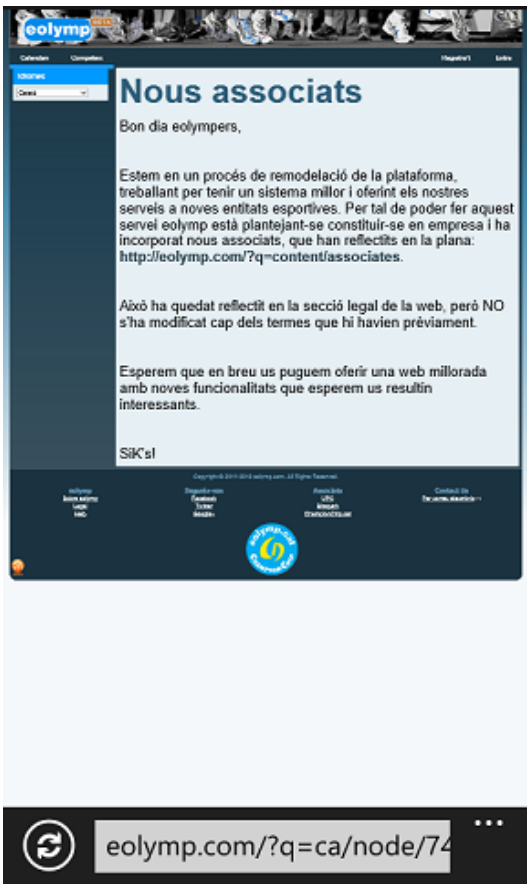


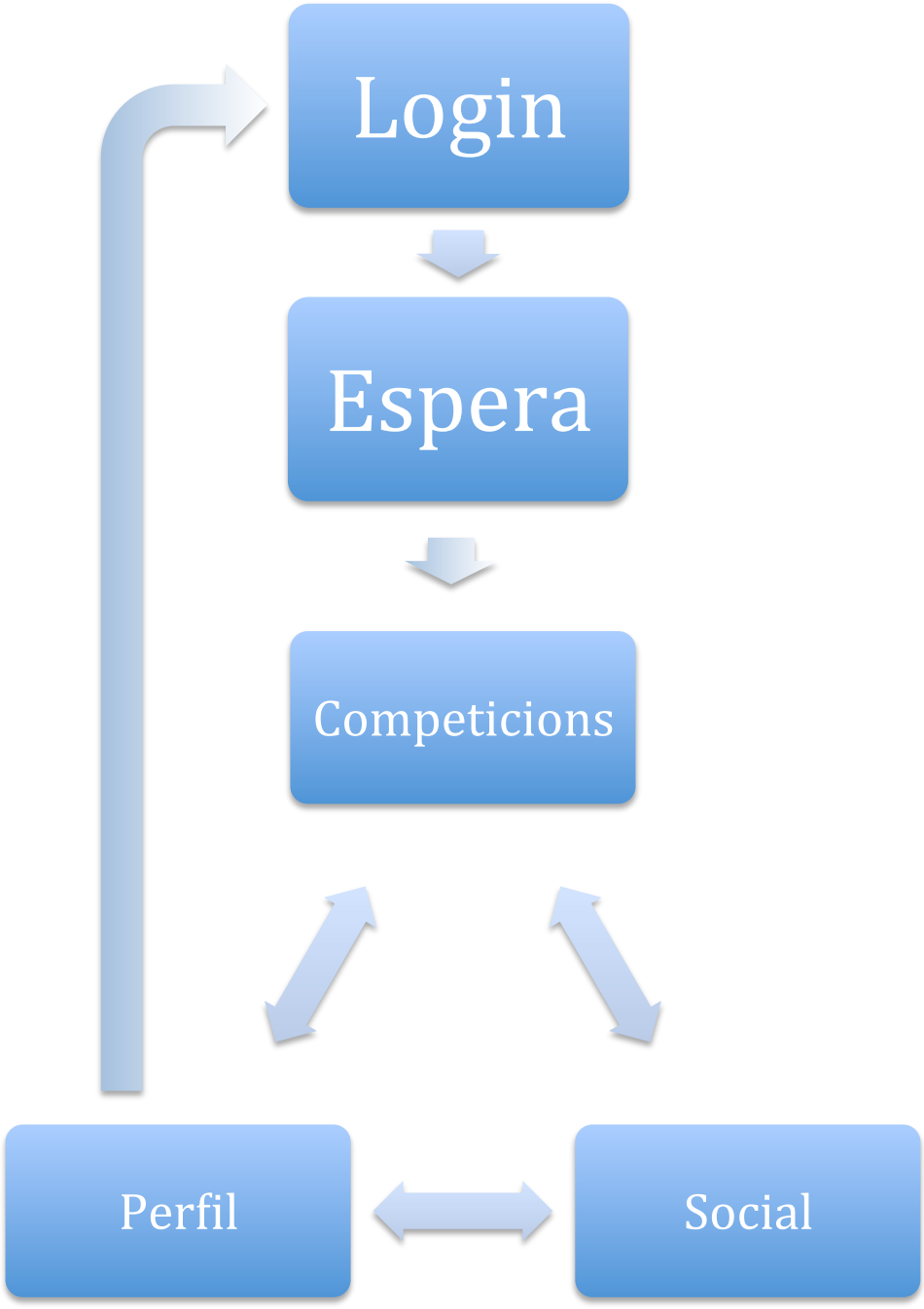
Figura 5.6 Detall social

Si tornem a l'aplicació, la següent pantalla que ens trobem es la del perfil. Aquí hi trobem les dades del nostre usuari juntament amb la nostra foto de perfil. Aquesta pantalla és totalment informativa, a excepció del botó que hi ha a la part inferior esquerra, que serveix per esborrar les dades locals de l'aplicació i tornar a la pantalla de login inicial (*Delete my data*).

Figura 5.7 Pantalla perfil

5.2 Diagrama de funcionament

El diagrama de flux de l'aplicació, per tant, és el següent:



5.3 Estructura del codi

A causa de l'acord de confidencialitat amb la UPC, per tal de no difondre el codi, aquest no s'inclourà en el document, però si en mostrarem fragments, per poder comprendre el funcionament de la seva estructura a grans trets.

La solució completa, que inclou l'aplicació d'escriptori i la versió per a Windows Phone ja comentada amb anterioritat, compta amb una carpeta compartida on hi ha el codi comú per a totes elles anomenada SH_eolymp. Aquesta carpeta fa servir la utilitat Externals de Subversion, que ens permet replicar parts del codi a diferents projectes i actualitzar cada un d'ells en el moment que ho desitgem, de manera que per exemple podem fer-hi els canvis desitjats per a iPhone sense que això afecti a la versió d'escriptori fins al moment en que funcioni correctament i actualitzem els fitxers locals.

5.3.1 SH_eolymp

Dins d'aquesta carpeta hi trobem, entre d'altres, els següents fitxers principals:

- ConnexioDrupalXMLRPC.cs

  Dins d'aquest fitxer hi ha les definicions dels mètodes XMLRPC del servidor: system.connect, user.login i views.get.

  També aquesta classe emmagatzema una col·lecció amb les competicions de l'usuari i una altra amb l'apartat social.

  Per últim, hi ha definides les funcions que ens permeten interactuar amb el servidor i són accessibles des de la resta del codi. Funcions com ara *UserLogin*, que permet autentificar a l'usuari contra el servidor, *guardarNodeActivitats*, que guarda una activitat realitzada per l'usuari al servidor i *getPictureUser* ,que obté la foto de perfil de l'usuari.

- Funcions.cs

  Conté funcions genèriques accessibles per a la resta del codi i que poden ser d'utilitat en qualsevol moment, com per exemple *mostrarMissatgeOK*, que mostra un missatge per pantalla amb un botó per acceptar-lo.

- SportsUsuari.cs

  Classe sense contingut que només conté l'estructura d'un esport.

5.3.2 Codi específic per a Windows Phone

Els fitxers que hi ha fora d'aquesta carpeta però dins del projecte de l'aplicació per a Windows Phone són específics per aquesta plataforma. A dins, hi trobem tant el disseny de les interfícies gràfiques d'usuari com el funcionament general de l'aplicació. Destacarem les següents classes principals:

- App.xaml

  Classe principal de l'aplicació Windows Phone. A dins hi guardem els recursos que es faran servir a la resta de l'aplicació, com ara una instància pública a la classe ConnexioDrupalXMLRPC i una altra a la classe XmlRpcStruct, de manera que durant tota la vida de la aplicació només fem servir un únic objecte.

- Login.xaml

  Conté el disseny gràfic de la pantalla de login i la seva programació, que gestiona l'autentificació de l'usuari fent servir els mètodes anteriorment descrits de la classe *ConnexioDrupalXMLRPC*. Seguidament, dóna pas a la següent pantalla d'espera.

- Waiting.xaml

  Conté el disseny gràfic de la pantalla d'espera.

- MainPage.xaml

  Conté el disseny en forma de panorama de les tres pantalles principals, Competicions, Social i Perfil, així com la seva programació i el seu comportament.

- LocalizedStrings.cs, AppResources.resx, AppResources.es-CA.resx, AppResources.es-ES.resx, AppResources.it-IT.resx

  Aquests fitxers permeten disposar de l'aplicació en els idiomes català, castellà, italià i anglès (idioma per defecte) mitjançant la definició d'etiquetes amb les seves descripcions en l'idioma concret.

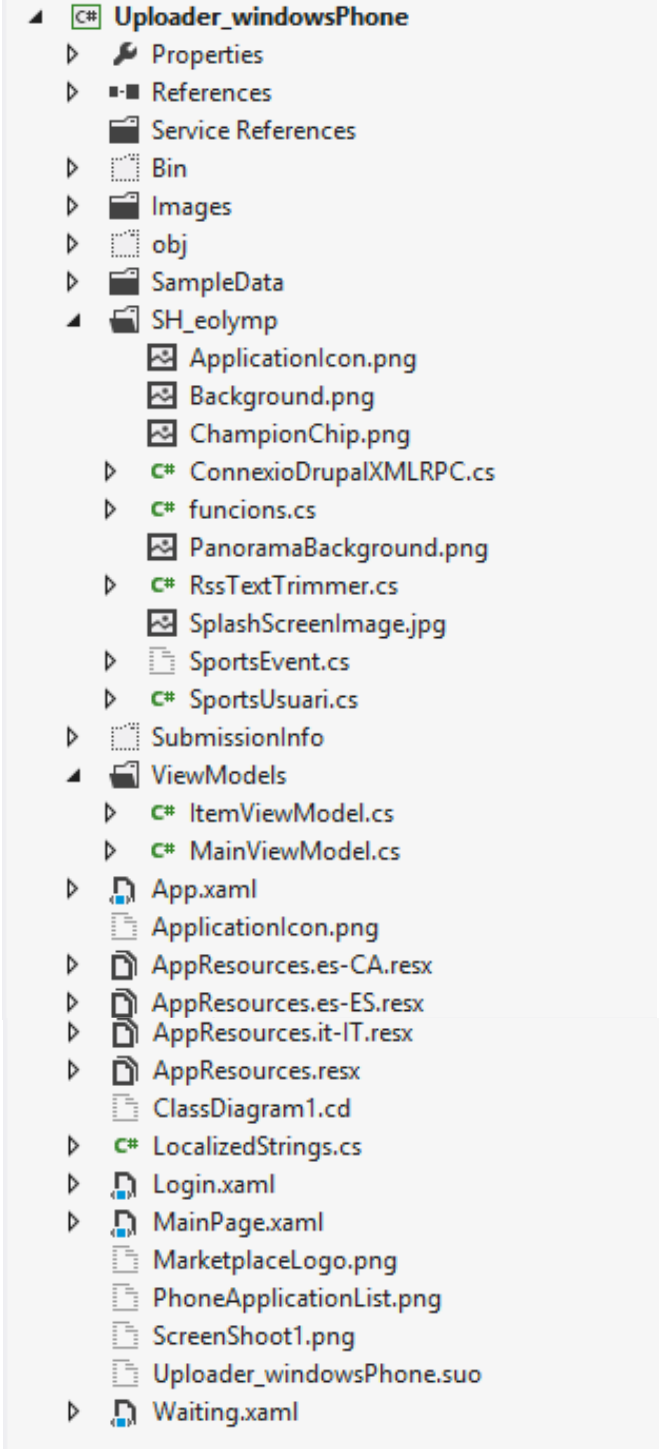


Figura 5.8 Estructura inicial aplicació Windows Phone

6 Conceptes de programació

Abans d'endinsar-nos amb detall en l'aplicació final resultant per iOS cal esmentar alguns conceptes que seran claus per entendre la distribució del codi font: el patró MVC i la gestió d'interfícies gràfiques en iOS, ambdós relacionats entre ells.

6.1 El patró MVC

MVC són les sigles de Model, Vista, Controlador i fan referència a un patró a tenir en compte a l'hora de programar qualsevol aplicació en general i aplicacions per iOS en particular.

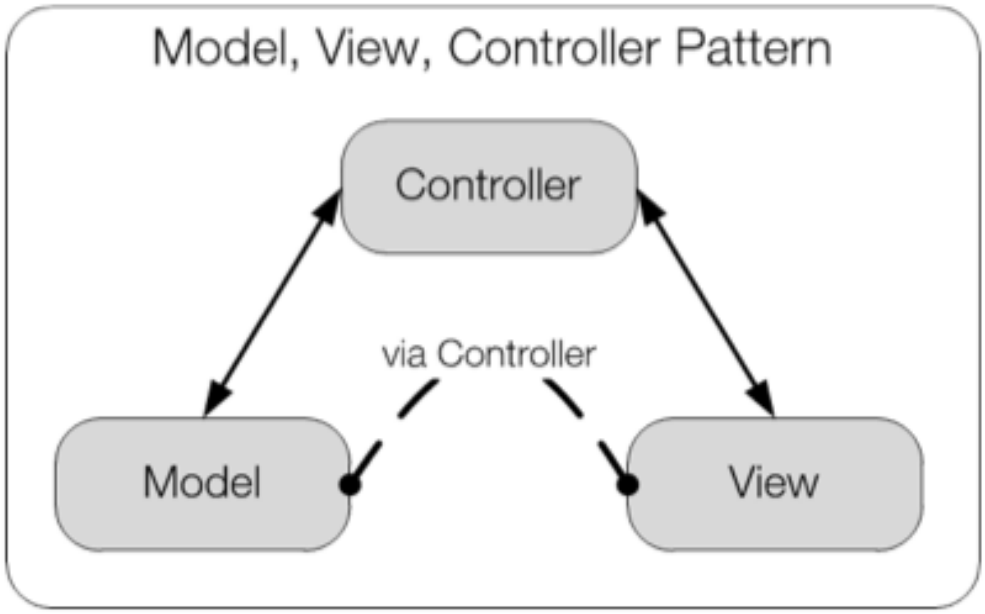


Figura 6.1 Esquema del patró Model, Vista, Controlador

Aquest patró d'arquitectura de software separa les dades, la lògica d'una aplicació i la interfície gràfica de l'usuari en tres components diferenciats, que són el model, la vista i el controlador.

Seguint aquest esquema un model pot tenir diferents vistes, cada una d'elles amb el seu corresponent controlador.

Aquest patró de disseny es basa en les idees de reutilització de codi i la separació de conceptes, característiques que busquen facilitar la tasca de desenvolupament i el seu posterior manteniment.

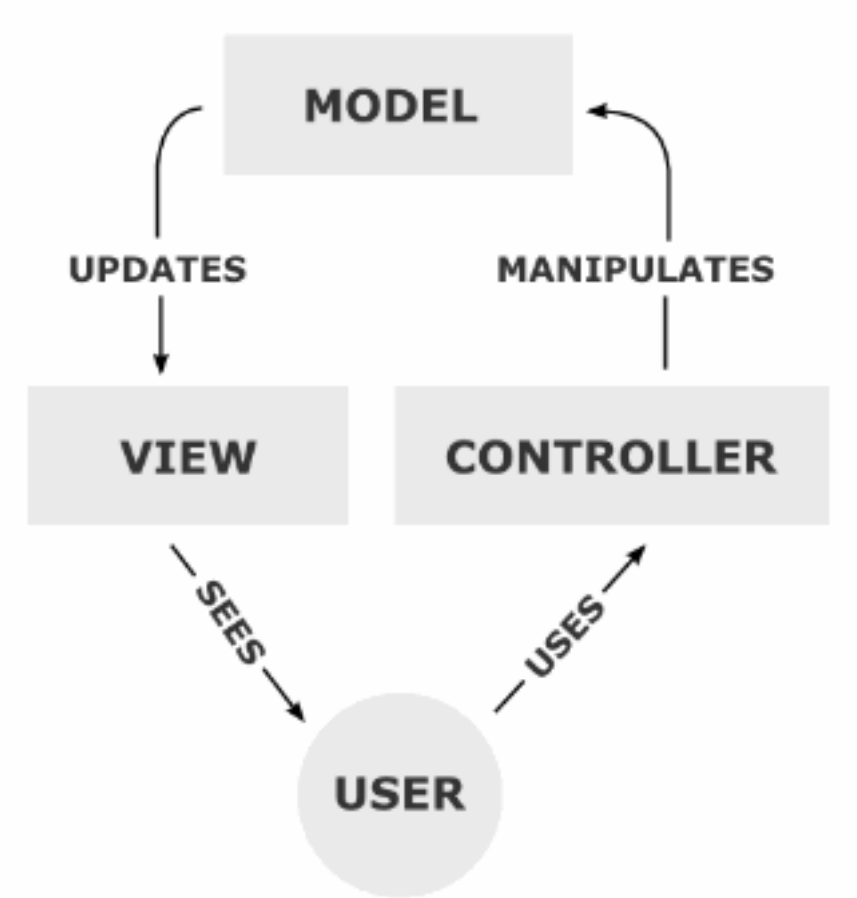


Figura 6.2 Detall del patró MVC

Podem descriure els components del patró de la següent manera:

Model

És la representació específica de la informació amb la qual el sistema treballa. La lògica de dades assegura la integritat d'aquestes i permet derivar noves dades. Això vol dir que aquí s'operen les dades i les regles de negoci associades al sistema, incloent l'anàlisi sintàctic i el processament de les dades d'entrada i de sortida.

El model és el responsable de:

- Accedir a la capa d'emmagatzematge de les dades. L'ideal és que el model sigui independent del sistema d'emmagatzematge.
- Defineix la funcionalitat del sistema.
- Porta un registre de les vistes i els controladors del sistema.
- Si estem davant d'un model actiu, aquest notificarà a les vistes els canvis que un agent extern pugi produir en elles.

Vista

La vista representa el model, normalment la interfície d'usuari. La vista és la capa de l'aplicació que veu l'usuari en un format adequat per interactuar.

Són les responsable de:

- Rebre dades del model i mostrar-les a l'usuari.
- Tenen un registre del seu controlador associat (normalment perquè és la vista qui fa la instància al controlador).
- Poden donar servei "d'actualització" perquè sigui invocat pel controlador o pel model.

Controlador

El controlador és la capa que controla tot el que pot realitzar la nostra aplicació. Respon a esdeveniments, generalment accions de l'usuari i invoca canvis en el model i probablement a la vista. Està format per accions que es representen amb funcions en una classe.

És el responsable de:

- Rep els esdeveniments d'entrada
- Conté regles de gestió d'esdeveniments, del tipus "SI event X llavors Acció Y". Aquestes accions poden suposar peticions al model o a les vistes.

6.2 Gestió d'interfícies gràfiques d'usuari en iOS

Una interfície gràfica d'usuari és aquell element que visualment permet presentar a l'usuari la informació d'una manera còmoda i eficaç i afegeix interoperabilitat amb el programa.

En el cas concret dels telèfons intel·ligents, a causa de la seva mida limitada, cal ser creatiu a l'hora de presentar la informació per pantalla. La forma més usual de procedir és dividir i agrupar la informació en diferents pantalles (vistes o *views*), mostrar d'inici la vista principal i, segons l'usuari va interaccionant amb l'aplicació, la informació apareix o desapareix canviant de vista.

Dins del sistema operatiu iOS hi ha diverses opcions de fer aquesta tasca i en aquest capítol explicarem les que s'han fet servir en aquesta aplicació.

6.2.1 View Controller

Els objectes View Controller proporcionen la infraestructura i el control necessaris per tal de gestionar una vista i de fer reaccionar l'aplicació en conseqüència a les accions de l'usuari. Dit d'una altra manera, és la capa situada entre la pantalla de l'aparell i la vista mostrada en aquell moment i s'encarrega de controlar aquesta última.

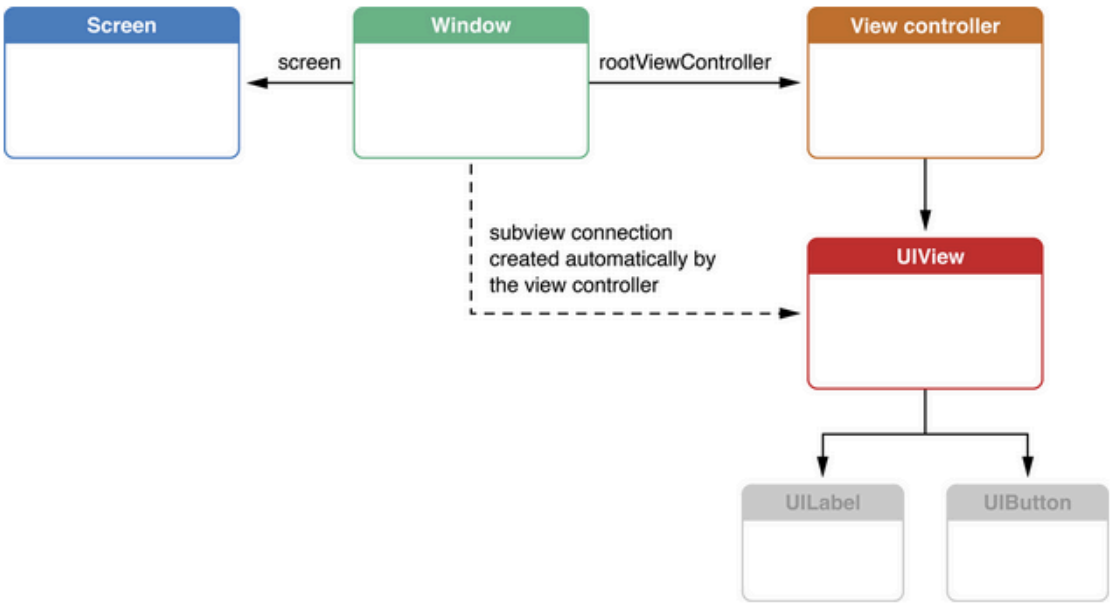


Figura 6.3 Esquema d'un View Controller

6.2.2 Tab Bar Controller

Els objectes Tab Bar Controllers o controladors de barra de pestanyes són una manera molt pràctica a l'hora de gestionar una aplicació que conté més d'una vista.

Es tracta d'un objecte de tipus contenidor al que se li afegeixen controladors de vista i les presenta en una barra a la part inferior de la pantalla amb tantes pestanyes com vistes s'ha afegit.

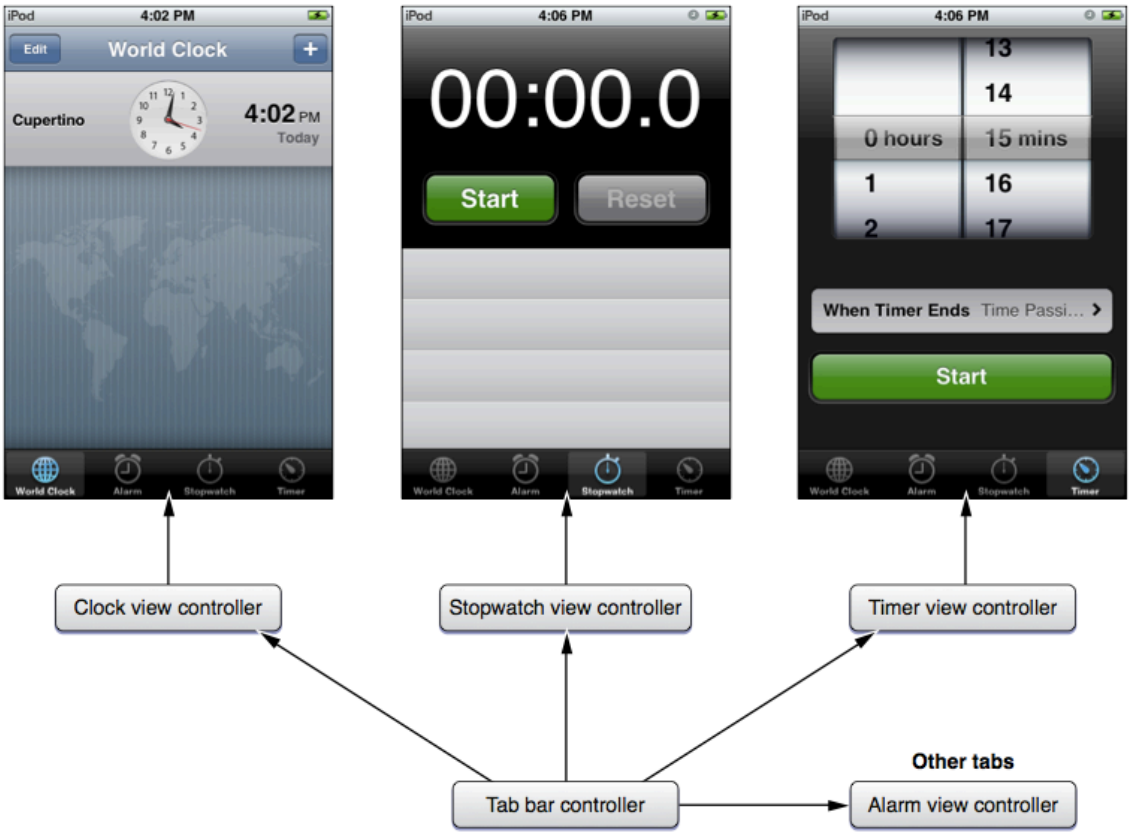


Figura 6.4 Exemple d'un Tab Bar Controller

D'aquesta manera, tal i com es pot veure a l'exemple de la Figura 6.4, l'usuari pot alternar ràpidament entre les diferents vistes de l'aplicació i saber en tot moment quina és la vista actual i quines són les altres disponibles.

6.2.3 Table View Source

Moltes aplicacions necessiten mostrar dades de manera tabulada i és per aquesta raó que existeixen diferents eines per a treballar amb taules en iOS.

Una d'elles, i que farem servir més endavant, és la classe *UITableViewSource*. La seva funció és proporcionar contingut i comportament personalitzat a les cel·les d'una taula. En realitat treballen com a model de dades ja que emmagatzemen les cel·les i les proporciona a la taula quan aquesta ho requereix.

Per tal de realitzar una classe personalitzada que sigui herència de UITableViewSource, aquesta ha de tenir com a mínim dos mètodes definits:

- RowsInSection: retorna el número total de files que la taula ha de mostrar.
- GetCell: retorna una cel·la plena de contingut i el seu índex corresponent.

```
1  public class TableSource : UITableViewSource {
2      string[] tableItems;
3      string cellIdentifier = "TableCell";
4      public TableSource (string[] items)
5      {
6          tableItems = items;
7      }
8      public override int RowsInSection (UITableView tableview, int section)
9      {
10         return tableItems.Length;
11     }
12     public override UITableViewCell GetCell (UITableView tableView, MonoTouch.Founda
13     {
14         UITableViewCell cell = tableView.DequeueReusableCell (cellIdentifier);
15         // if there are no cells to reuse, create a new one
16         if (cell == null)
17             cell = new UITableViewCell (UITableViewCellStyle.Default, cellIdentifier
18         cell.TextLabel.Text = tableItems[indexPath.Row];
19         return cell;
20     }
21 }
```

Figura 6.5 Exemple d'un TableViewSource senzill

7 L'aplicació per iPhone

A causa de l'acord de confidencialitat amb la UPC no podem mostrar el codi de l'aplicació en aquest document. És per això que crearem una aplicació base com a exemple fent servir les mateixes eines i característiques emprades en la solució real, tot i que d'una manera molt més esquemàtica, i mostrant part del resultat final real.

7.1 Creació del projecte

El primer pas seria crear el projecte, que anomenarem AplicacioSimple, amb el Xamarin Studio, tot indicant que es tracta d'una aplicació per iOS del tipus *Single View Application*:

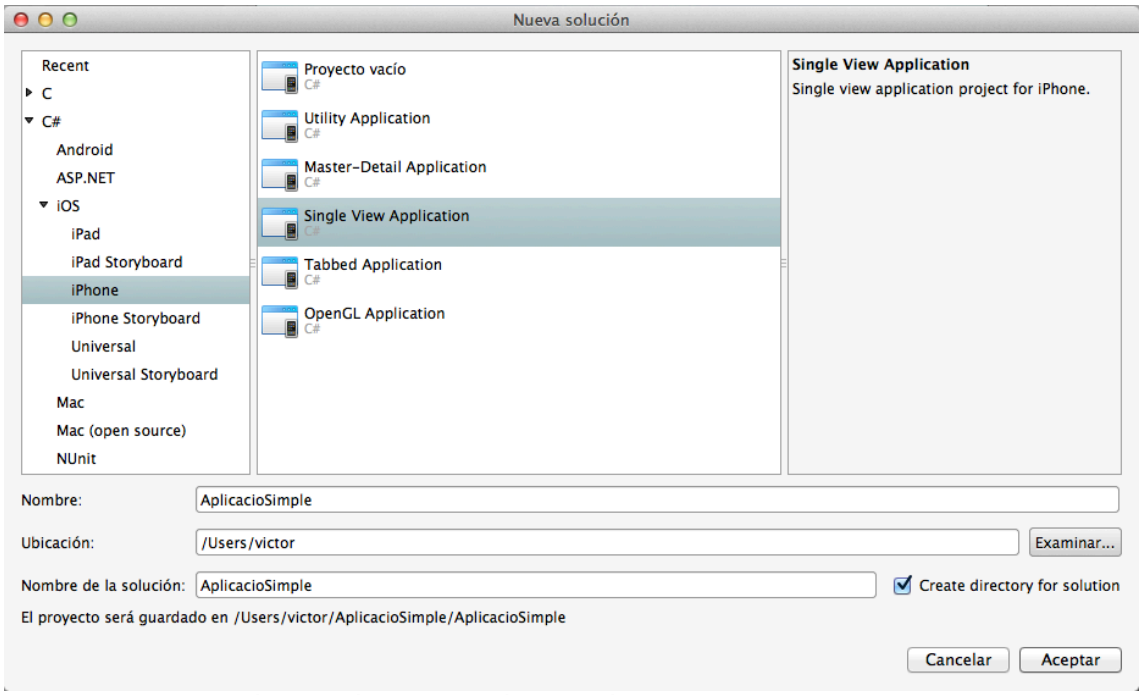


Figura 7.1 Creació d'una aplicació simple per iPhone

Un cop hem especificat el tipus, nom i ubicació dels fitxers del projecte el mateix Xamarin Studio ens crea els fitxers base de l'aplicació estructurats de la següent manera:

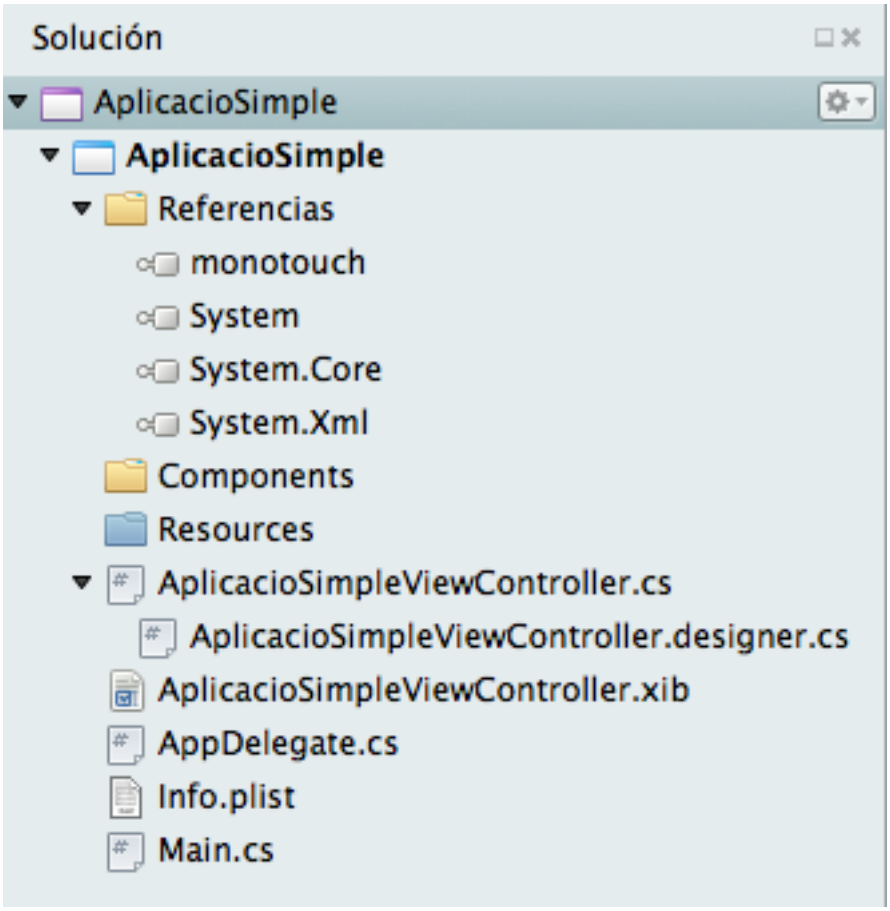


Figura 7.2 Estructura inicial d'una aplicació per iPhone

Com podem observar a la Figura 7.2, ens ha creat una solució i un projecte dins d'aquesta amb el nom que hem especificat en el pas anterior, AplicacioSimple.

Dins hi trobem la carpeta "Referencias", la qual conté les llibreries mínimes necessàries per executar l'aplicació. Aquí destacarem la llibreria *monotouch*, que és la que ens ofereix Xamarin i la que fa possible la posterior transformació del codi C# en codi natiu per iOS.

El sistema ens crea per defecte dues carpetes més, Components i Resources, perquè si volem afegir més components o recursos que farà servir el nostre programa siguem acurats i fem servir aquestes per guardar-ho.
En aquest cas no afegirem res més i per tant restaran buides.

L'editor també ha generat diferents fitxers específics i necessaris per a l'entorn iPhone:

- Main.cs: és la classe principal de l'aplicació. S'encarrega de cridar a la següent classe, AppDelegate, i hi podem incloure mètodes o variables que podran ser utilitzades des de qualsevol part de l'aplicació.

- AppDelegate.cs: és la classe responsable de cridar per primer cop la interfície gràfica d'usuari, en el nostre cas la pantalla de Login.

- Info.plist: serveix per definir les característiques de l'aplicació, com ara el nom, la versió o fins i tot la icona que veurà l'usuari final al seu telèfon.

7.2 Disseny de les vistes

Al haver creat una aplicació del tipus Single View Applicaction el sistema ens ha creat una vista amb el seu controlador. Aquesta s'organitza en tres fitxers, relacionats entre ells pel nom principal *AplicacioSimpleViewController* però amb diferents extensions. Aquesta és la tríada que es genera per a cada pantalla que té la nostra aplicació i que formen un *View Controller*, concepte explicat al capítol 6.2.1.

La funció de cada un d'aquests fitxers és la següent:

- AplicacioSimpleViewController.xib

És el fitxer on està tot el disseny gràfic de la vista. Fent-hi doble clic ens obrirà el dissenyador d'interfícies de Xcode per tal que puguem afegir tots els controls necessaris a la vista d'una manera totalment visual i modificar les seves propietats segons convingui.

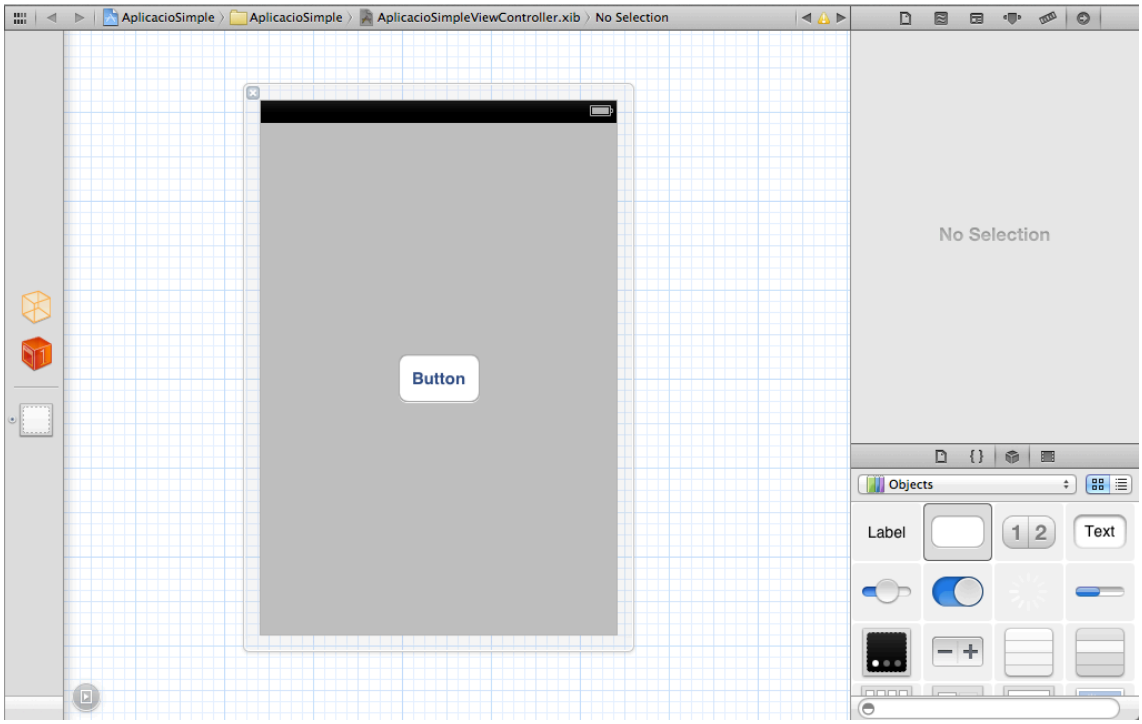
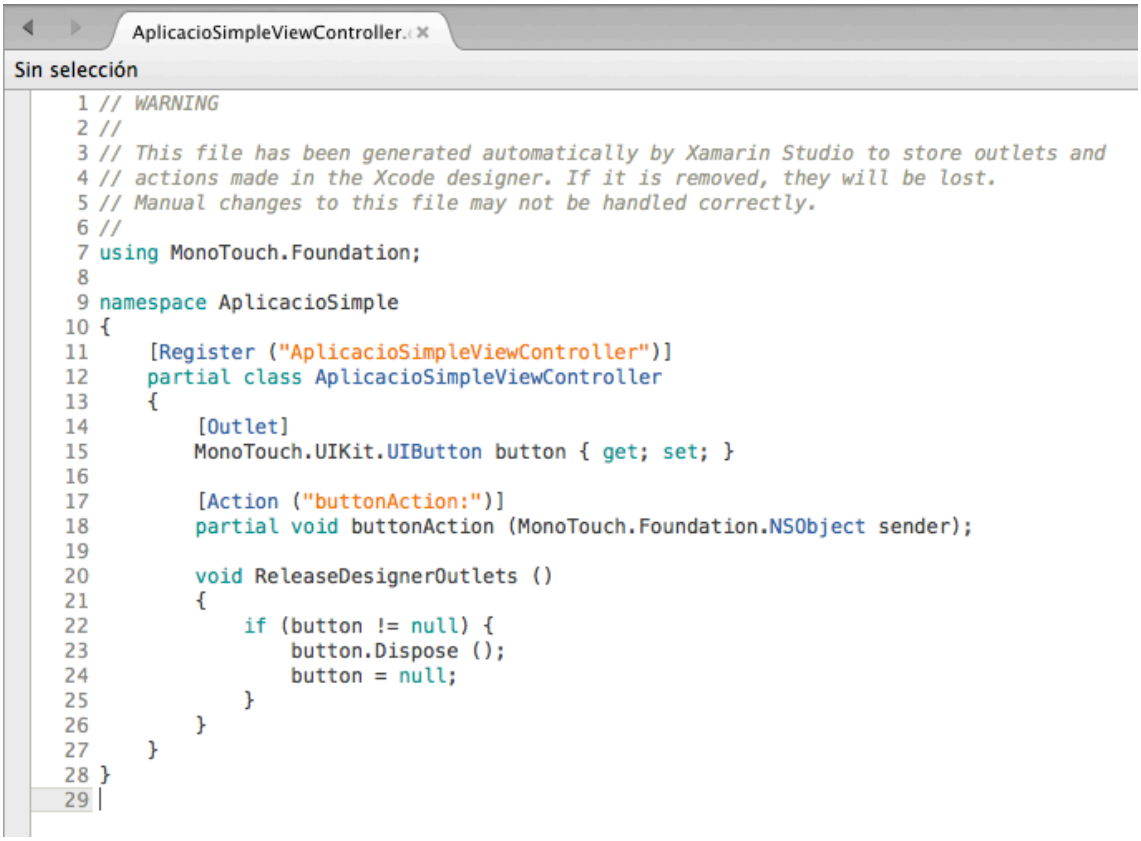


Figura 7.3 Detall del disseny del fitxer .xib

Inicialment tenim una pantalla buida a la que podem anar afegint els elements visuals desitjats des de la barra d'objectes que tenim a la dreta. S'ha afegit un botó d'exemple.

- AplicacióSimpleViewController.designer.cs

En aquest fitxer hi trobem la definició de tots els outlets que haguem definit a través de Xcode. Els outlets, com ja hem vist al capítol 3.1, fan possible l'accés des de dins del codi als objectes i accions de les vistes i ens permetràn interactuar amb els elements de pantalla.




```
1  // WARNING
2  //
3  // This file has been generated automatically by Xamarin Studio to store outlets and
4  // actions made in the Xcode designer. If it is removed, they will be lost.
5  // Manual changes to this file may not be handled correctly.
6  //
7  using MonoTouch.Foundation;
8
9  namespace AplicacioSimple
10 {
11     [Register ("AplicacioSimpleViewController")]
12     partial class AplicacioSimpleViewController
13     {
14         [Outlet]
15         MonoTouch.UIKit.UIButton button { get; set; }
16
17         [Action ("buttonAction:")]
18         partial void buttonAction (MonoTouch.Foundation.NSObject sender);
19
20         void ReleaseDesignerOutlets ()
21         {
22             if (button != null) {
23                 button.Dispose ();
24                 button = null;
25             }
26         }
27     }
28 }
29
```

Figura 7.4 Detall del contingut del fitxer .designer.cs

Per fer l'exemple hem creat dos outlets senzills a botó a través del dissenyador de Xcode: un per poder accedir a les propietats del botó (button), i un altre per poder accedir a les accions provocades pel botó i poder actuar en conseqüència (buttonAction).

- AplicacioSimpleViewController.cs

Dins hi trobem la programació necessària per tal que la vista es carregui i funcioni correctament. El codi inicial és auto generat per Xcode però hi podem afegir codi personalitzat.
A l'exemple següent fem que quan el botó sigui polsat el text d'aquest canviï per "botó polsat".

AplicacioSimpleViewController.

AplicacioSimpleViewController ▸ Sin selección

```
1  using System;
2  using System.Drawing;
3  using MonoTouch.Foundation;
4  using MonoTouch.UIKit;
5
6  namespace AplicacioSimple
7  {
8      public partial class AplicacioSimpleViewController : UIViewController
9      {
10         public AplicacioSimpleViewController () : base ("AplicacioSimpleViewController", null)
11         {
12         }
13
14         public override void DidReceiveMemoryWarning ()
15         {
16             // Releases the view if it doesn't have a superview.
17             base.DidReceiveMemoryWarning ();
18
19             // Release any cached data, images, etc that aren't in use.
20         }
21
22         public override void ViewDidLoad ()
23         {
24             base.ViewDidLoad ();
25
26             // Perform any additional setup after loading the view, typically from a nib.
27
28         }
29
30         public override bool ShouldAutorotateToInterfaceOrientation (UIInterfaceOrientation toInterfaceOrientation)
31         {
32             // Return true for supported orientations
33             return (toInterfaceOrientation != UIInterfaceOrientation.PortraitUpsideDown);
34         }
35
36         partial void buttonAction (NSObject sender)
37         {
38             button.SetTitle("botó polsat", UIControlState.Normal);
39         }
40     }
41 }
42
```

Figura 7.5 Detall del fitxer .cs

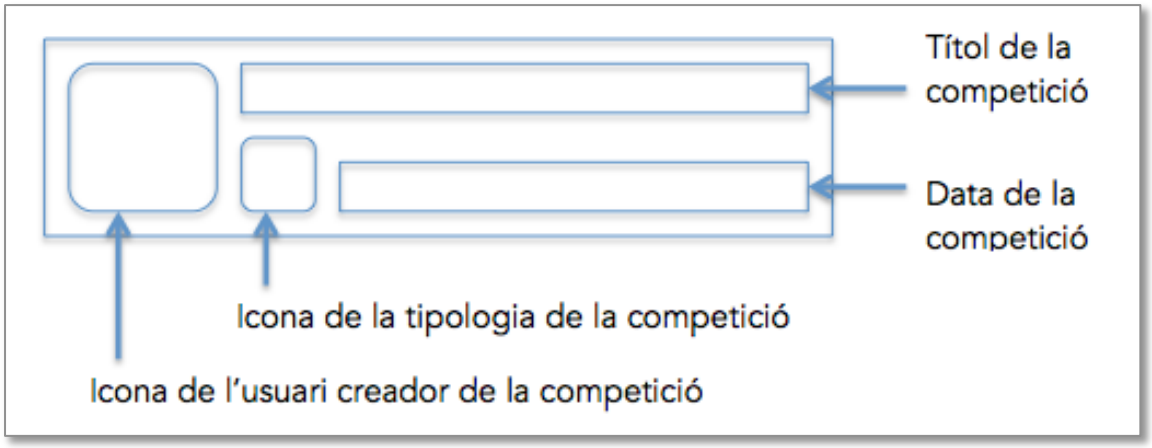
Amb totes aquestes eines ja podem començar a donar forma a l'aplicació, dissenyant totes les vistes necessàries, definint el comportament dels objectes que contenen i creant els outlets necessaris.

7.3 Personalització de les taules

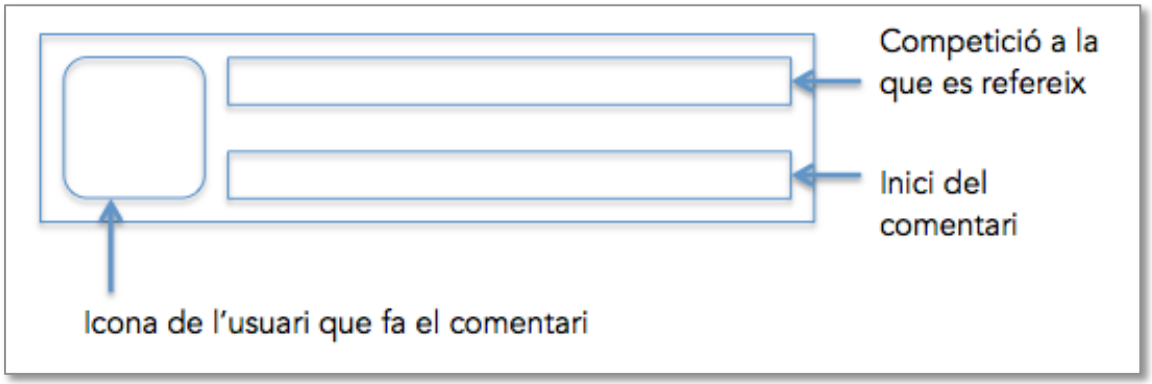
Per tal de mostrar el contingut de les pantalles de competicions i social farem ús de les taules. iOS proporciona alguns models ja fets i que en la majoria de casos s'adapten al que el programador necessita. En el nostre cas i per tal d'assimilar-se el màxim possible a l'aplicació per a Windows Phone, haurem de personalitzar una mica aquests models estàndard.

Observant les pantalles inicials de l'aplicació (capítol 5.1) per a Windows Phone podem extreure el disseny de com han de ser les cel·les de cadascuna de les taules.

Les cel·les de la taula de competicions ha de tenir el següent format:



I les cel·les de la taula social han de seguir el següent esquema:



Per tal de programar el disseny i el comportament dels dos tipus de cel·les s'han creat dos fitxers, TableSourceCompete.cs i TableSourceSocial.cs, tots dos descendents de la subclasse *UITableViewSource* (capítol 6.2.3).

Una subclasse UITableViewSource (font de la taula) és assignada a cada UITableView (vista de la taula). La vista de la taula consulta la seva classe font per determinar com renderitzar-se a sí mateixa (per exemple, quantes files són necessàries i l'alçada de cada una d'elles, si són diferents de les cel·les per defecte). I el més important, la font proporciona a cada cel·la el seu contingut.

Per tal que les cel·les tinguin el disseny i el comportament requerit cal definir tres mètodes (els dos primer obligatòriament):

- RowsInSection: retorna el número total de files que la taula ha de mostrar.

- GetCell: retorna una cel·la plena de contingut i el seu índex corresponent.

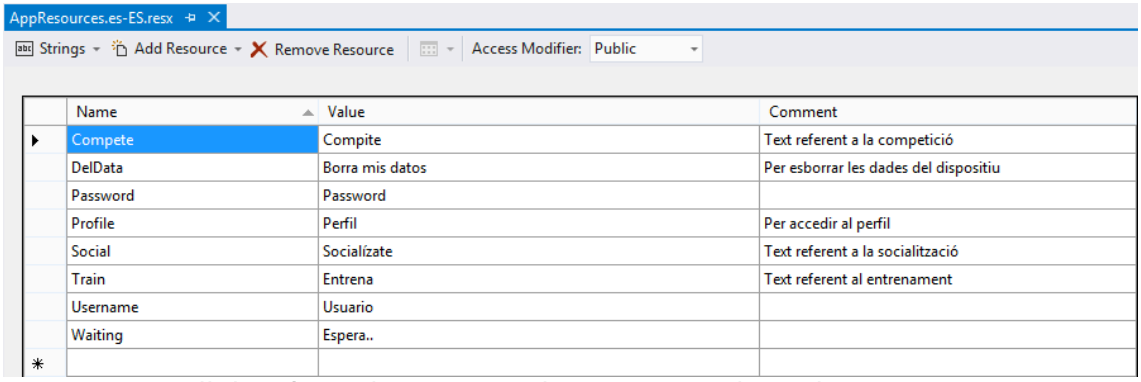
- RowSelected: dins podem especificar l'acció que succeeix al seleccionar una cel·la (en el nostre cas l'aplicació ens redirigeix al navegador i ens mostra el web del comentari o competició seleccionat).

Dins de cada un dels fitxers creats per nodrir les taules, a banda dels mètodes obligatoris, s'ha definit una estructura (*Struct*) amb el contingut necessari de cada cel·la i una col·lecció per tal de poder emmagatzemar totes les cel·les de la taula.

7.4 Internacionalitzant l'aplicació

En l'actualitat, les aplicacions tant per a dispositius mòbils com per a sistemes fixes d'escriptori, es fan pensant en un abast més global que local i és per això que les eines de desenvolupament compten amb un sistema, per tal que el programador no hagi de fer diferents versions del programa només per haver de canviar l'idioma en que aquesta es mostra a l'usuari.

El sistema de Windows Phone compta amb uns fitxers anomenats AppResources per aconseguir aquesta funcionalitat i podem tenir tants com idiomes vulguem que tingui la nostra aplicació. La forma en que es fa és especificant dins de cada fitxer unes paraules clau i la seva traducció en l'idioma que estem contemplant, de manera que des del codi enlloc de fer servir una paraula exacta farem servir aquesta paraula clau, i segons la nostra localització, el sistema mostrarà la traducció que hem definit per aquesta paraula clau en l'idioma correcte.



	Name	Value	Comment
▸	Compete	Compite	Text referent a la competició
	DelData	Borra mis datos	Per esborrar les dades del dispositiu
	Password	Password	
	Profile	Perfil	Per accedir al perfil
	Social	Socialízate	Text referent a la socialització
	Train	Entrena	Text referent al entrenament
	Username	Usuario	
	Waiting	Espera..	
*			

Figura 7.6 Detall d'un fitxer d'internacionalització en Windows Phone

A la Figura 7.6 veiem el detall d'un d'aquests fitxers en l'entorn Windows Phone. Aquest en concret és el que conté l'idioma castellà i es veu com s'hi defineixen paraules clau i la seva traducció en l'idioma actual. Per especificar l'idioma que conté el fitxer es fa servir l'extensió d'aquest, de manera que per tal de disposar de l'aplicació de Windows Phone en diferents idiomes comptem amb els següents fitxers:

- AppResources.es-ES.resx: idioma castellà
- AppResources.es-CA.resx: idioma català
- AppResources.it-IT.resx: idioma italià
- AppResources.resx: idioma per defecte, en aquest cas anglès

Per replicar el mateix funcionament en l'entorn iOS farem ús de les carpetes .lproj. Es tracta de definir tantes carpetes .lproj com idiomes vulguem que tingui la nostra aplicació i afegir el codi de l'idioma davant de .lproj. Dins de la carpeta haurem d'afegir un fitxer de text pla amb nom *Localizable.strings* i que contingui una parella clau-valor per a cada paraula específica en l'idioma desitjat.

Figura 7.7 Detall d'un fitxer d'internacionalització en iPhone

De la mateixa manera que hem fet per a la plataforma Windows Phone haurem d'especificar les següents carpetes amb els següents idiomes:

- es.lproj: idioma castellà
- ca.lproj: idioma català
- it.lproj: idioma italià
- en.lproj: en aquest cas l'idioma per defecte es l'anglès

7.5 Els controladors

Fins ara hem dissenyat totes les vistes de l'aplicació i som capaços d'omplir-les amb el contingut corresponent però encara ens falta alguna classe capaç de gestionar quan toca mostrar cadascuna d'elles; aquesta serà el controlador.

En el nostre cas s'ha optat per separar la classe controladora en dues parts:

- ControladorLogin: s'encarrega de gestionar l'autenticació amb el servidor mitjançant la pantalla de Login, mostra la pantalla Waiting mentre fa el procés i després ens redirigeix a les pantalles principals de l'aplicació si hem tingut èxit o retorna a la pantalla inicial de Login en cas contrari.

- ControladorPrincipal: s'encarrega de gestionar les vistes principals de l'aplicació, que són Compete, Social i Profile, a més de gestionar la recepció del servidor de les dades que hauran de tenir aquestes.

Aquestes dues classes, a banda de gestionar el comportament general de l'aplicació per iPhone, compleixen una funció molt important dins de la solució global i és que també aglutinen la major part del codi compartit entre plataformes.

En termes generals, a l'aplicació per a Windows Phone la gestió de l'autenticació es realitzava dins de la mateixa classe Login.xaml i la recepció de les dades i comportament genèric de les vistes principals era dins del fitxer MainClass.xaml (veure capítol 5.3.2).

Amb la creació d'aquests dos nous controladors, que formen part de la carpeta SH_eolymp ja que són comuns a la resta de plataformes, s'ha aconseguit extreure d'aquests dos fitxers .xaml propis de Windows Phone la major part possible que no forma part específica del disseny de les vistes. D'aquesta manera el comportament de l'aplicació a totes les plataformes queda definit aquí i queda separat del disseny gràfic propi de cada plataforma.

Aquesta reunificació de codi en aquests dos fitxers controladors no hauria estat possible sense una característica molt important del llenguatge C#, que són les directrius de compilació condicional.

Al capítol 3.3 s'ha introduït el tema però arribats aquest punt és necessari aprofundir una mica més en l'ús que se n'ha fet en aquest projecte.

Per a l'aplicació d'iPhone s'ha definit l'etiqueta MONOTOUCH (a l'inici del projecte es va definir aquest nom d'etiqueta per el nom de la llibreria de Mono per iPhone i, quan aquesta va canviar el nom per Xamarin.iOS, es va decidir mantenir el mateix nom d'etiqueta).

Això comporta que cada vegada que al codi trobem l'expressió **#if MONOTOUCH** el compilador tindrà en compte la part del codi que hi sigui dins i obviarà la que sigui dins del #else.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Shapes;
using Uploader;
using CookComputing.XmlRpc;
using System.ComponentModel;
#if MONOTOUCH
using MonoTouch.Foundation;
using MonoTouch.UIKit;
using Uploader_iphone;
#else
using Microsoft.Phone.Controls;
using System.IO.IsolatedStorage;
using System.Windows.Threading;
using System.Windows.Navigation;
#endif
```

Figura 7.9 Exemple de compilació condicional iPhone

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Shapes;
using Uploader;
using CookComputing.XmlRpc;
using System.ComponentModel;
#if MONOTOUCH
using MonoTouch.Foundation;
using MonoTouch.UIKit;
using Uploader_iphone;
#else
using Microsoft.Phone.Controls;
using System.IO.IsolatedStorage;
using System.Windows.Threading;
using System.Windows.Navigation;
#endif
```

Figura 7.8 Exemple de compilació condicional Windows Phone

Un clar exemple el trobem a les figures superiors. Aquestes captures de pantalla són part del ControladorLogin.cs i és el punt on es defineixen les llibreries que es faran servir.

Des de la primera línia fins que apareix la instrucció #if MONOTOUCH formaran part de totes les plataformes, ja que no hi ha cap condicional.

En el cas de l'iPhone, després de #if MONOTOUCH el compilador només tindrà en compte les següents línies fins trobar un #else (també podria ser un #endif) a la plataforma que tingui especificada l'etiqueta MONOTOUCH (en aquest cas la d'iPhone), i que són les següents tres línies.

Les línies compreses entre el #else i el #endif no seran tingudes en compte pel compilador a la plataforma iPhone i per tant no formaran part de la seva aplicació.

Un cop tancat el condicional amb #endif el compilador tornarà al seu mode normal i no farà cap mena de distinció entre plataformes i el codi serà comú a totes elles sense excepció.

Si comparem aquesta mateixa captura de pantalla feta des de l'entorn d'iPhone contra la mateixa part de codi però en l'entorn Windows Phone visualment s'apreciarà la diferència, ja que les parts no fetes servir a cada plataforma apareixen com si fos un comentari i el programa les marca amb un color diferent (normalment el gris).

Podríem classificar tots aquests condicionals que s'han fet servir en aquest projecte en tres tipus:

- Condicionals per excloure/incloure llibreries

Aquest cas és clar i és el que hem vist a l'exemple anterior. D'aquesta manera podem excloure llibreries que són exclusives i necessàries per iOS en el codi Windows Phone i viceversa, sense haver-nos de preocupar per la compatibilitat d'aquestes en l'altra plataforma i el mateix per quan es vulgui afegir Android.

- Condicionals segons la manera de procedir

Tot i la unificació de codi que s'ha dut a terme, dins dels dos controladors hi trobarem mètodes que no poden ser exactament iguals a totes les plataformes. Per això, dins d'alguns mètodes s'ha hagut d'especificar una manera concreta de procedir per a cada plataforma. Per exemple a l'hora d'executar tasques en segon pla el sistema difereix entre Windows Phone i iPhone.

```
#if MONOTOUCH
                UIApplication.SharedApplication.BeginInvokeOnMainThread(delegate
                {
                    m_Connect.m_errLogin = false;
                    Uploader_iphone.Application.window.RootViewController=new Login();
                });
#else
                dispatcher.BeginInvoke(delegate()
                 {
                    m_Connect.m_errLogin = false;
                    ((PhoneApplicationFrame)Application.Current.RootVisual).RemoveBackEntry();
                    ((PhoneApplicationFrame)Application.Current.RootVisual).Navigate(new Uri("/Login.xaml"
                });
#endif
```

Figura 7.10 Fragment de codi amb compilació condicional

A la Figura 7.10 veiem un d'aquests casos. Si observem la part de Windows Phone, que apareix en gris, hi trobem la figura del *dispatcher*. Aquest objecte proporciona la capacitat d'executar porcions de codi en un subprocés diferent a l'actual i de manera asíncrona amb la instrucció *BeginInvoke*. En la plataforma iPhone no existeix aquest objecte dispatcher i és la pròpia aplicació

que actua com a tal. Per això proporciona la instrucció *BeginInvokeOnMainThread*. Fent servir aquesta obtenim el mateix comportament que a Windows Phone.

Tenint en compte aquest concepte queda clar llavors l'ús que haurem de fer de la compilació condicional; en qualsevol mètode que per a Windows Phone faci servir el *BeginInvoke* haurem d'afegir el condicional de que només ho faci a la seva plataforma, mentre que si estem a la plataforma iPhone faci el mateix però fent la crida a *BeginInvokeOnMainThread*.

- Condicionals segons l'estructura de l'aplicació

Com que l'estructura de les aplicacions no és exactament la mateixa, tot i que el funcionament sí que ho és, als controladors també s'ha hagut de fer servir els condicionals per especificar que certes parts del codi només han d'executar-se en el cas de l'iPhone.

Un exemple el trobem a l'hora de fer la crida i omplir els textos de les pantalles principals. A Windows Phone aquestes hi son programades en un únic fitxer (MainPage.xaml) ja que es fa servir el concepte de Panorama, mentre que en el cas de l'iPhone, al no disposar del mateix tipus de vista s'han creat les tres vistes en fitxers separats i és el controlador de pestanyes qui s'encarrega d'unificar-les. A conseqüència d'això, el codi d'iPhone haurà d'inicialitzar les tres pantalles, crear el controlador de pestanyes i incloure-hi les vistes, mentre que per a Windows Phone només cal cridar el fitxer .xaml.

```
#if MONOTOUCH
            //inicialitzem les tres pantalles i carreguem la primera d'elles al navigationController
            UIApplication.SharedApplication.BeginInvokeOnMainThread (delegate {
                compete=new Compete(competeHeader);
                social=new Social(socialHeader);
                profile=new Profile(this);
                train=new Train(trainHeader);
                UITabBarController tab=new UITabBarController();
                tab.ViewControllers = new UIViewController [] {compete, social, profile,train};
                Uploader_iphone.Application.window.RootViewController=tab;
            });

#endif
```

Figura 7.11 Fragment de codi que utilitza condicional per especificar codi iPhone

7.6 Pantalles de l'aplicació final

L'esquema de funcionament per a iPhone és el mateix que per a Windows Phone. També s'ha intentat fer les pantalles el més semblant possible, amb la

diferència que el control panorama de Windows Phone no existeix per a iPhone.

És per això que, un cop passat el login i la pantalla d'espera, a les tres pantalles principals, per canviar d'una a l'altra, s'ha optat, per raons de claredat, per una pantalla tabulada. És a dir, a la part inferior de la pantalla tindrem una pestanya per a cada una de les finestres disponibles i, per canviar d'una a l'altra, enlloc de desplaçar la pantalla, simplement pitgem a sobre de la pestanya de la finestra que volem obrir.

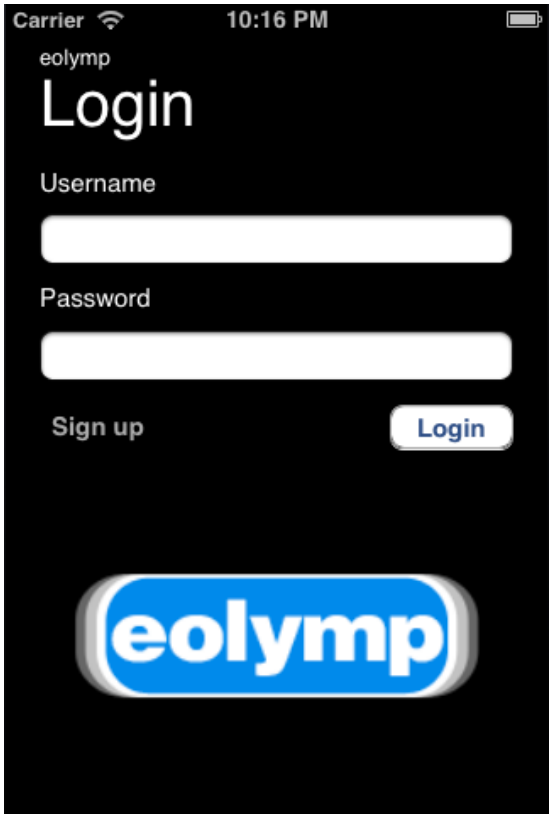


Figura 7.13 Pantalla de login iPhone

Figura 7.12 Pantalla d'espera iPhone

Figura 7.17 Pantalla competicions iPhone

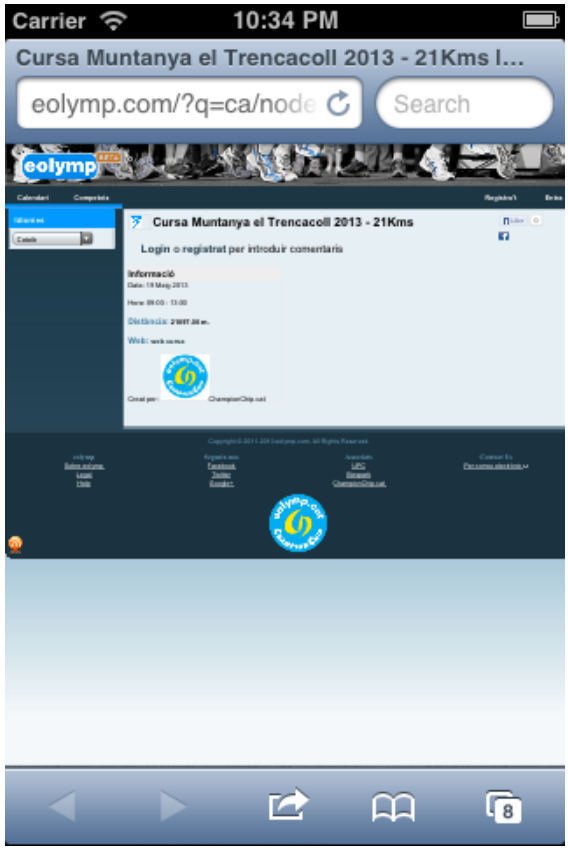


Figura 7.16 Detall competició iPhone

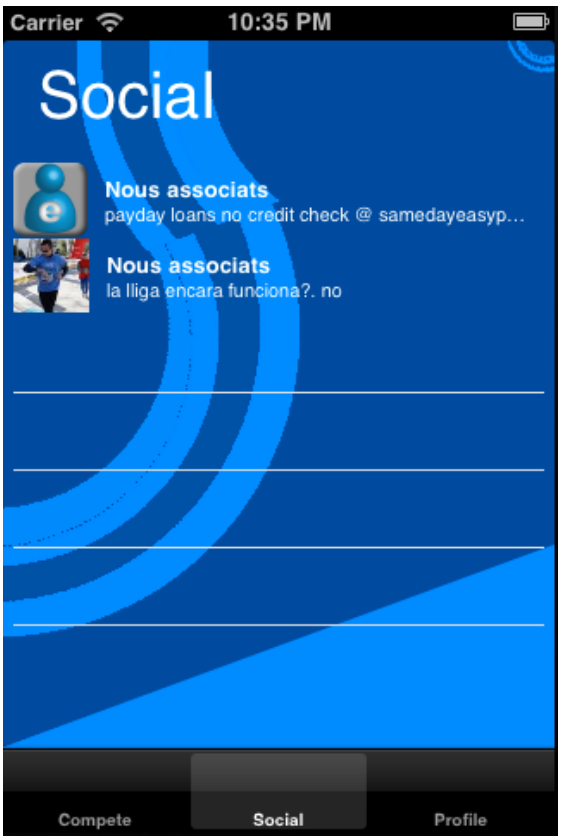


Figura 7.15 Pantalla social iPhone

Figura 7.14 Detall social iPhone

Figura 7.18 Pantalla perfil iPhone

7.7 Estructura del codi per iPhone

Amb tot el que hem vist fins ara ja ens trobem en disposició de poder muntar l'aplicació d'iPhone completa.

Un cop muntada, l'estructura de l'aplicació ens queda de la següent manera:

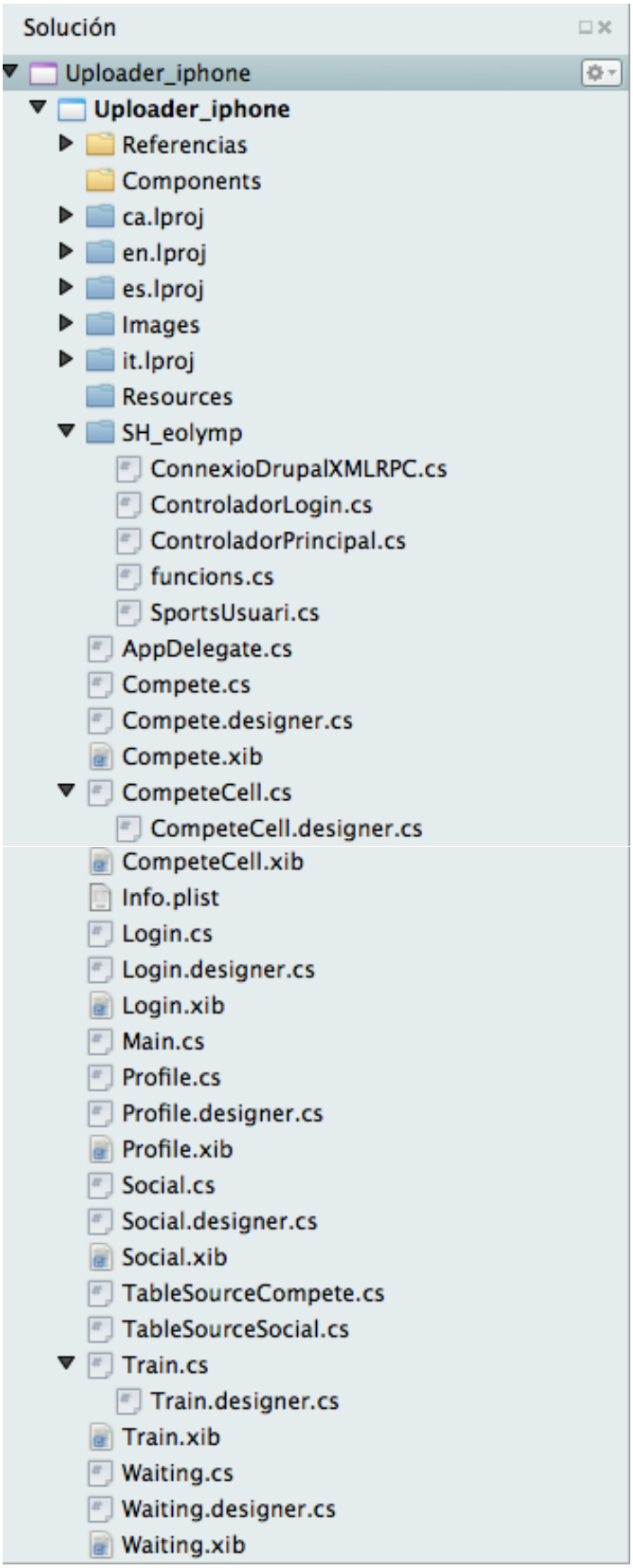


Figura 7.19 Estructura de l'aplicació iPhone

7.8 Reestructuració del codi

L'objectiu de la reestructuració del codi és integrar en l'estructura ja existent de fitxers l'aplicació per iPhone i reduir a la mínima expressió el codi específic de cada plataforma, de manera que el codi comú a totes les plataformes sigui el responsable del màxim possible de tasques.

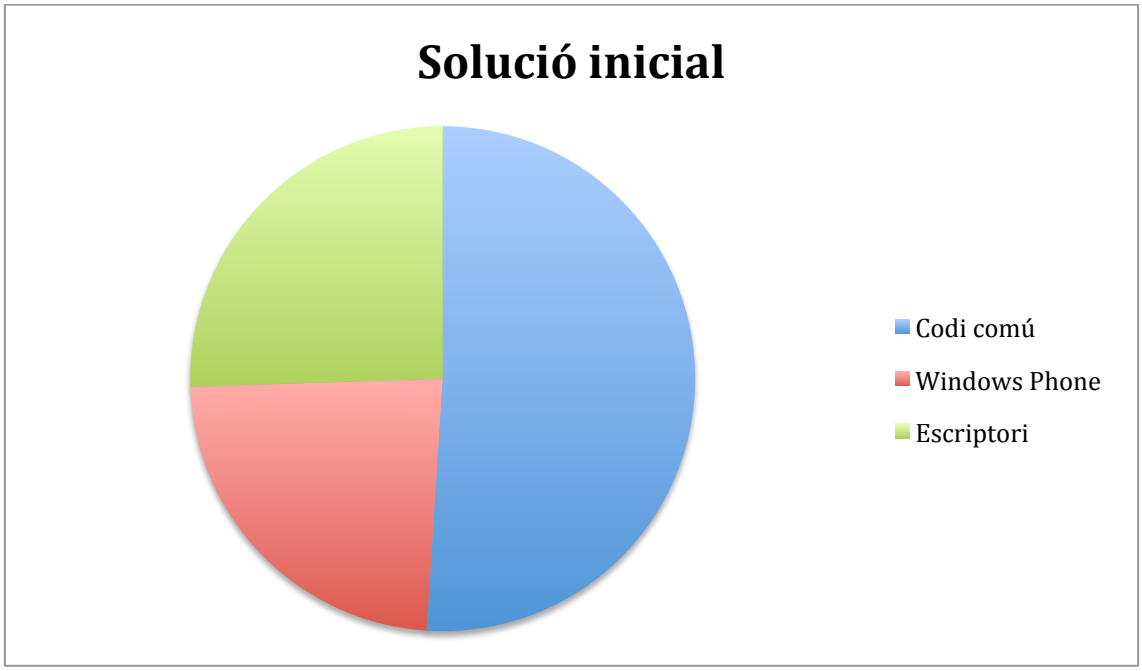


Figura 7.20 Distribució del codi abans de la reestructuració

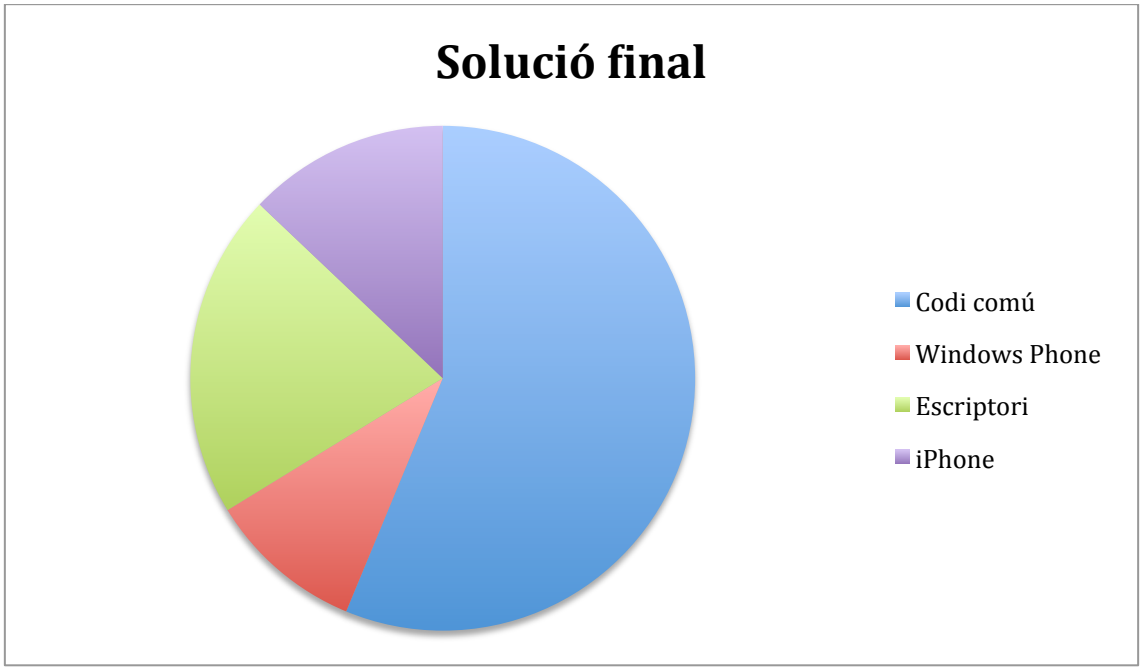


Figura 7.21 Distribució del codi després de la reestructuració

Si observem els dos gràfics comparatius anteriors, realitzats segons una aproximació de la mida dels fitxers resultants, podem veure com efectivament el codi comú ha augmentat de mida, ja que ara s'encarrega de més tasques, i com el codi específic per a Windows Phone ha minvat notablement. Observem també l'addició del nou codi per a iPhone.

La importància de l'augment del codi comú a totes les plataformes és clau a l'hora del posterior manteniment i ampliació de les aplicacions, ja que facilitarà les tasques al programador al estar agrupat i centralitzat.

Cal comentar també que l'aplicació d'escriptori s'ha volgut mantenir igual ja que el funcionament i l'aspecte difereixen molt de l'aplicació mòbil, i per tant, aquesta primera no farà ús dels controladors, només dels fitxers que estaven des d'un principi a la carpeta compartida (ConnexioDrupalXMLRPC, funcions i SportsUsuari).

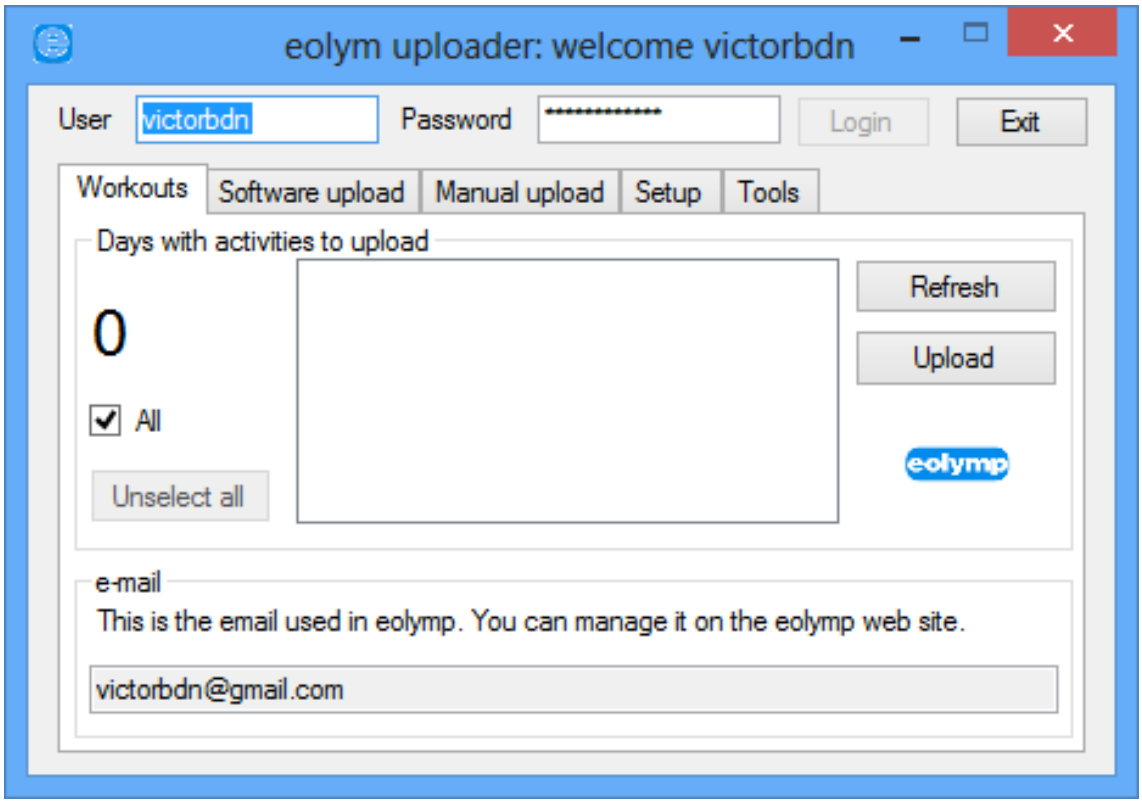
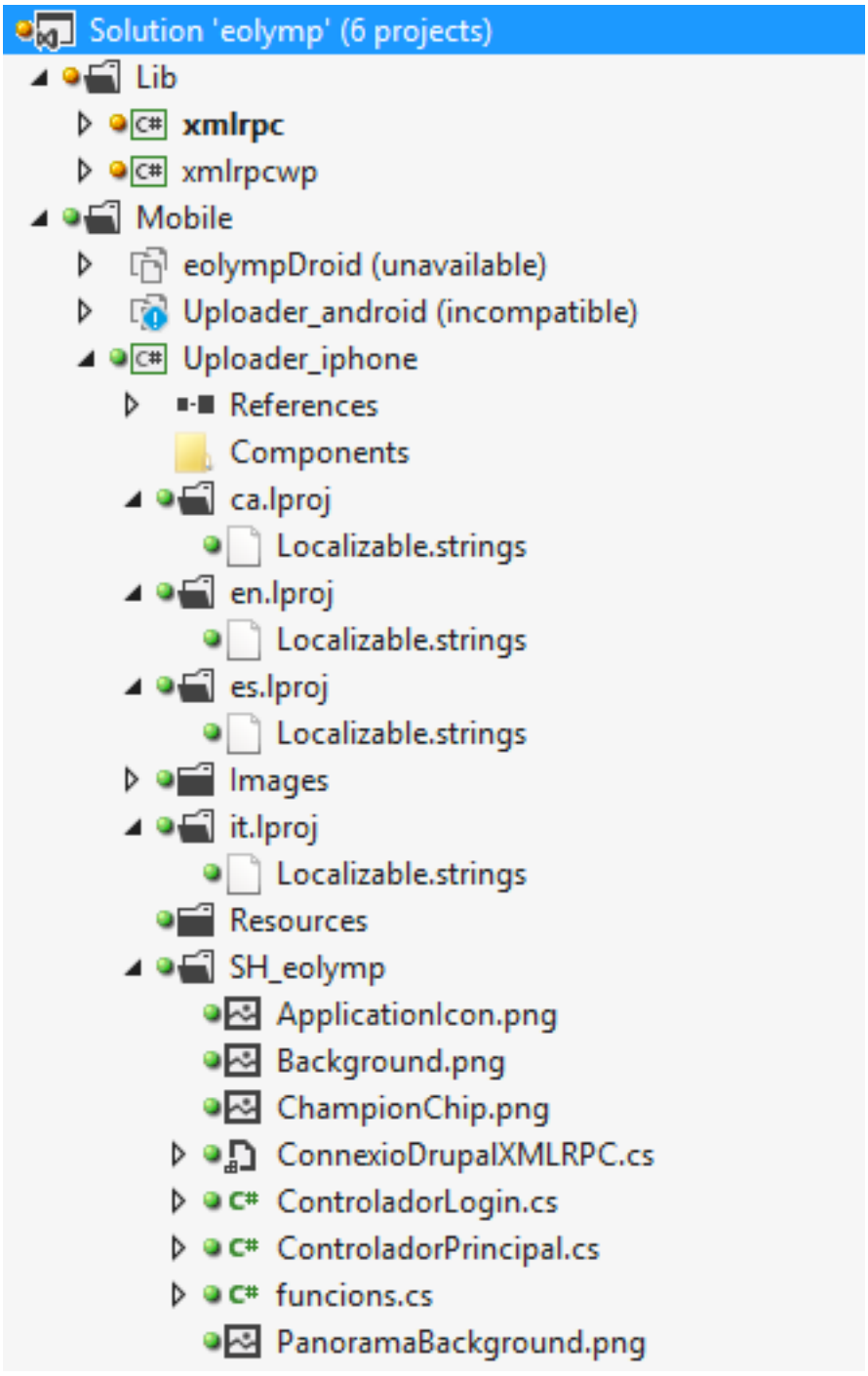


Figura 7.22 Captura de l'aplicació d'escriptori

L'aplicació per a iPhone s'ha desenvolupat mitjançant l'IDE proporcionat per Xamarin, perquè es va començar el projecte amb ell i perquè inicialment només aquí es podia executar el simulador d'iPhone. Com que la solució final està en l'entorn Visual Studio 2012 s'ha hagut d'incloure la d'iPhone en ella.

L'estructura final de la solució completa en Visual Studio 2012 és la següent:




RssTextTrimmer.cs
SplashScreenImage.jpg
SportsUsuari.cs
AppDelegate.cs
Compete.cs
Compete.designer.cs
Compete.xib
CompeteCell.cs
CompeteCell.designer.cs
CompeteCell.xib
Info.plist
Login.cs
Login.designer.cs
Login.xib
Main.cs
Profile.cs
Profile.designer.cs
Profile.xib
Social.cs
Social.designer.cs
Social.xib
TableSourceCompete.cs
TableSourceSocial.cs
Train.cs
Train.designer.cs
Train.xib
Waiting.cs
Waiting.designer.cs
Waiting.xib

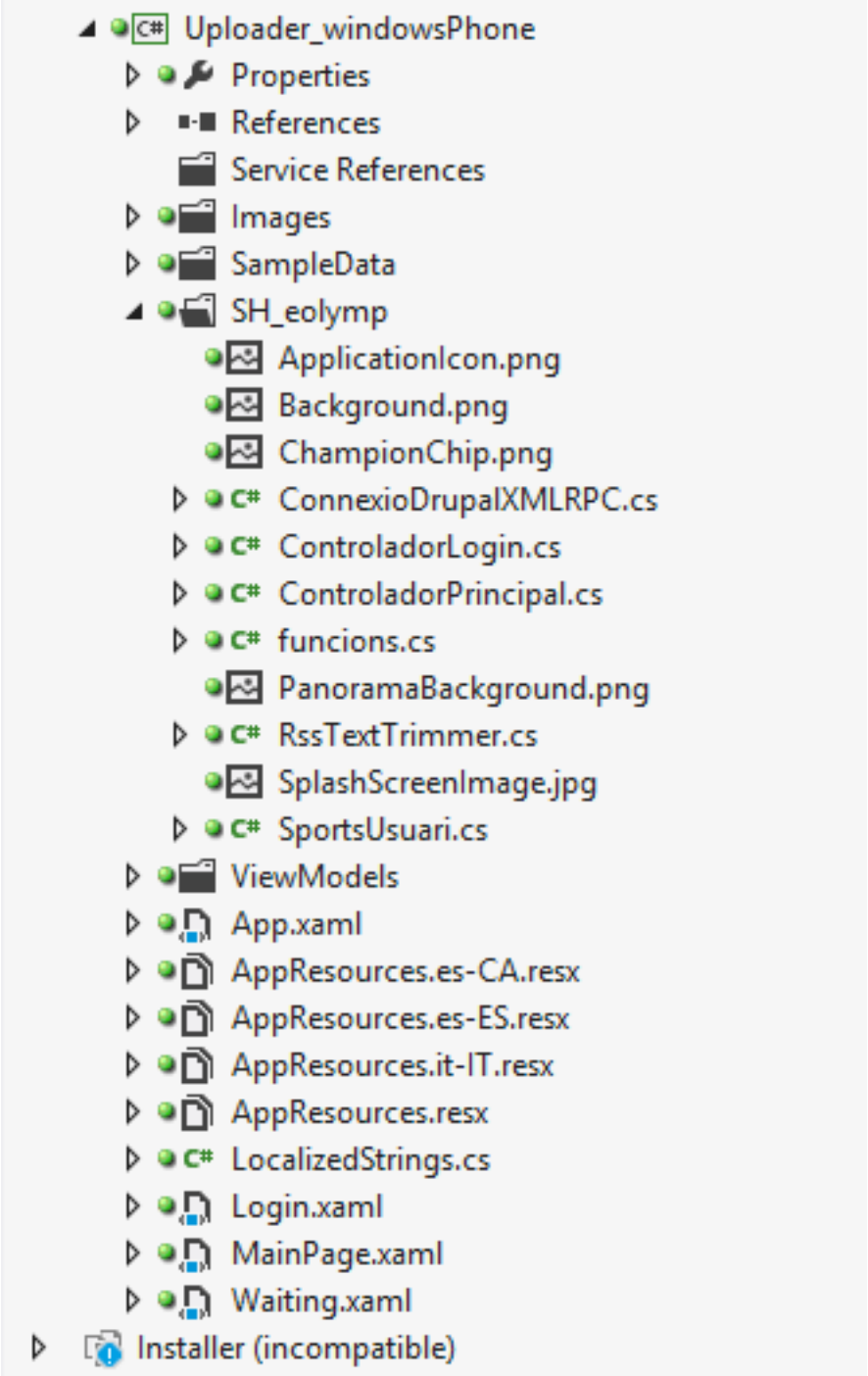

Uploader_windowsPhone
Properties
References
Service References
Images
SampleData
SH_eolymp
ApplicationIcon.png
Background.png
ChampionChip.png
ConnexioDrupalXMLRPC.cs
ControladorLogin.cs
ControladorPrincipal.cs
funcions.cs
PanoramaBackground.png
RssTextTrimmer.cs
SplashScreenImage.jpg
SportsUsuari.cs
ViewModels
App.xaml
AppResources.es-CA.resx
AppResources.es-ES.resx
AppResources.it-IT.resx
AppResources.resx
LocalizedStrings.cs
Login.xaml
MainPage.xaml
Waiting.xaml
Installer (incompatible)

Figura 7.23 Estructura de l'aplicació iPhone

8 Proves, problemes i solucions

8.1 Preparant l'entorn de proves

Com ja comentàvem al capítol 3.4, per aquest projecte s'ha fet servir Drupal. El servidor d'eolymp compta amb un servidor que l'incorpora però per no saturar-lo ni fer malbé cap dada s'ha optat per crear un servidor local que imiti el real, implementant en ell el mínim necessari.

Per a que Drupal funcioni ha d'estar rodant en un servidor web que disposi de base de dades MySQL i accepti el llenguatge PHP. A causa de les comoditats que ofereix i que és software gratuït s'ha optat pel programa MAMP. Aquest software inclou en un mateix paquet d'instal·lació tot el que necessitem per executar Drupal a l'equip:

- Mac: compatible amb el sistema operatiu Mac OS X.
- Apache: servidor web
- MySQL: sistema gestor de bases de dades
- PHP: llenguatge PHP

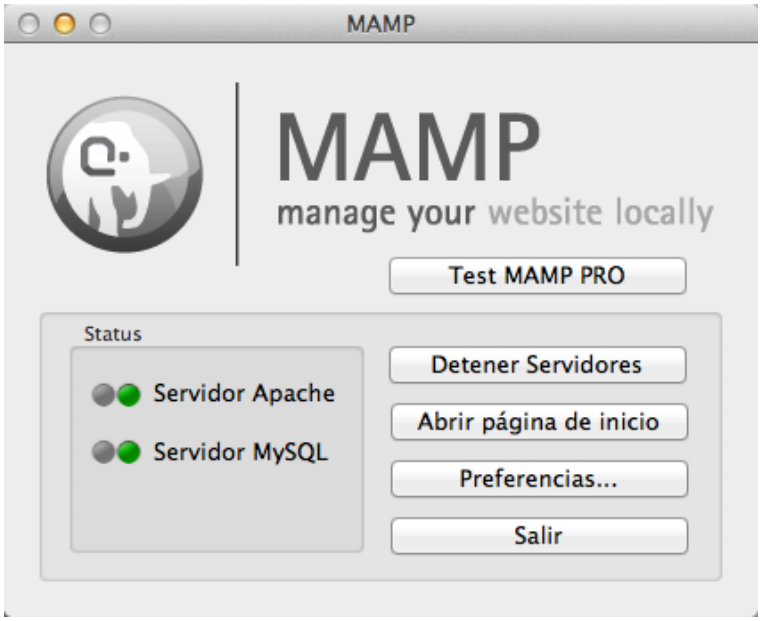


Figura 8.1 Captura de pantalla de MAMP

Un cop instal·lat i configurat correctament podem obrir-lo i iniciar els servidors per poder accedir-hi. Quan ho fem, si obrim la pàgina d'inici del nostre servidor local a través del navegador web tindrem accés a la configuració de la base de dades a través de phpMyAdmin.

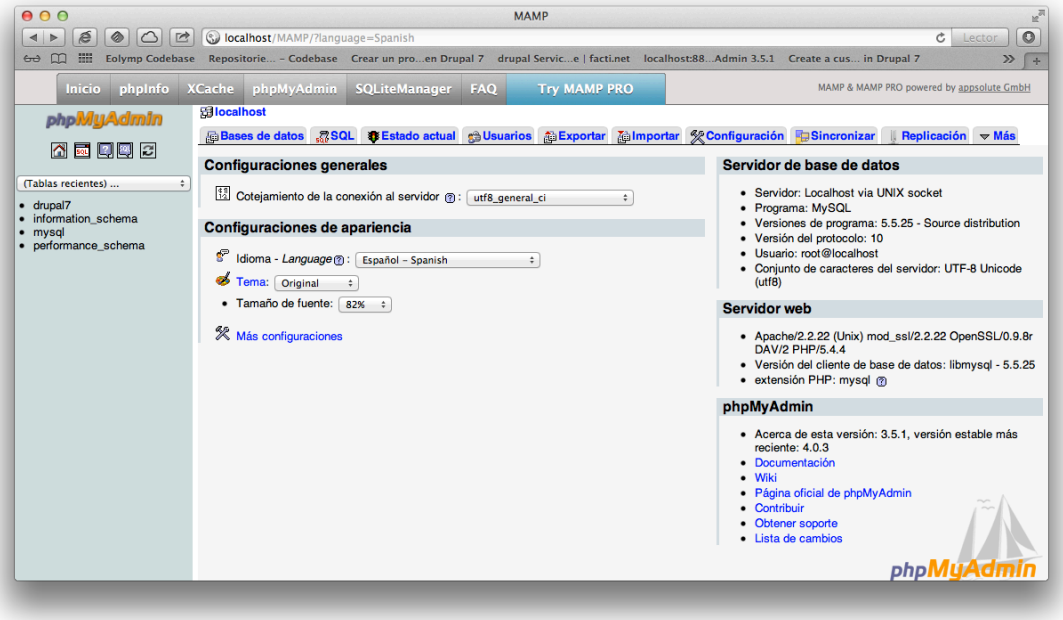


Figura 8.2 Captura de pantalla de phpMyAdmin

El phpMyAdmin és una eina que ens permet gestionar les bases de dades del nostre servidor a través d'una interfície web, i és d'aquesta manera que hem de crear una nova base de dades per Drupal; en aquest cas l'hem creada amb el nom *drupal7*. Només hem de crear-la ja que l'instal·lador de Drupal s'encarregarà d'afegir-hi el contingut que necessiti.

Ara ja podem procedir a la descarrega i instal·lació del Drupal. S'ha optat per la versió 7.19, que era l'última versió estable disponible en el moment de fer les proves. La instal·lació simplement requereix descarregar una carpeta i situar-la a l'arrel de la carpeta del servidor web. Fet això, si accedim a l'adreça del nostre servidor local ja ens apareix una pantalla de benvinguda de Drupal i ens insta a procedir amb la instal·lació, que bàsicament configura tot el necessari per al seu correcte funcionament.

Arribats a aquest punt ja tenim instal·lat Drupal al nostre servidor local i podem comprovar-ho accedit a la pàgina d'inici del servidor:

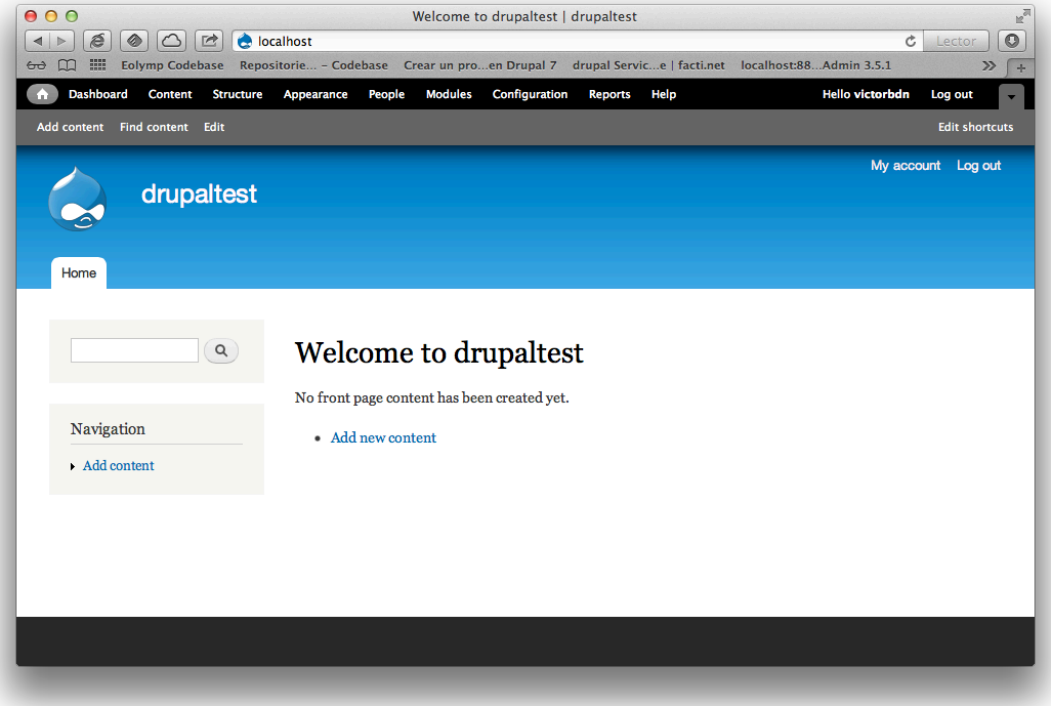


Figura 8.3 Pàgina d'inici de Drupal

El primer pas ara serà crear un nou compte d'usuari que ens permeti accedir a Drupal i configurar tot el necessari per fer les proves.

Tot seguit, necessitem habilitar el mòdul Services, que acostuma a venir instal·lat però desactivat, i el mòdul Node Service. Això ens permetrà comunicar amb el servidor.

Com que ens comunicarem amb ell mitjançant XML-RPC hem d'instal·lar-hi al nostre Drupal el mòdul XMLRPC Server i com a requisit d'aquest el mòdul Services. També ens farà falta instal·lar el mòdul Data.

Per procedir a instal·lar aquests mòduls, i qualsevol altre que necessitem, hem de descarregar-lo de l'apartat de mòduls de la web de Drupal i un cop tinguem els fitxers afegir-los mitjançant l'eina *Install new module*, que trobem a la secció de mòduls.

Amb tots aquest processos ja tenim llest el nostre servidor local Drupal per poder fer proves, i la primera que farem serà la de connectar la nostra aplicació amb ell.

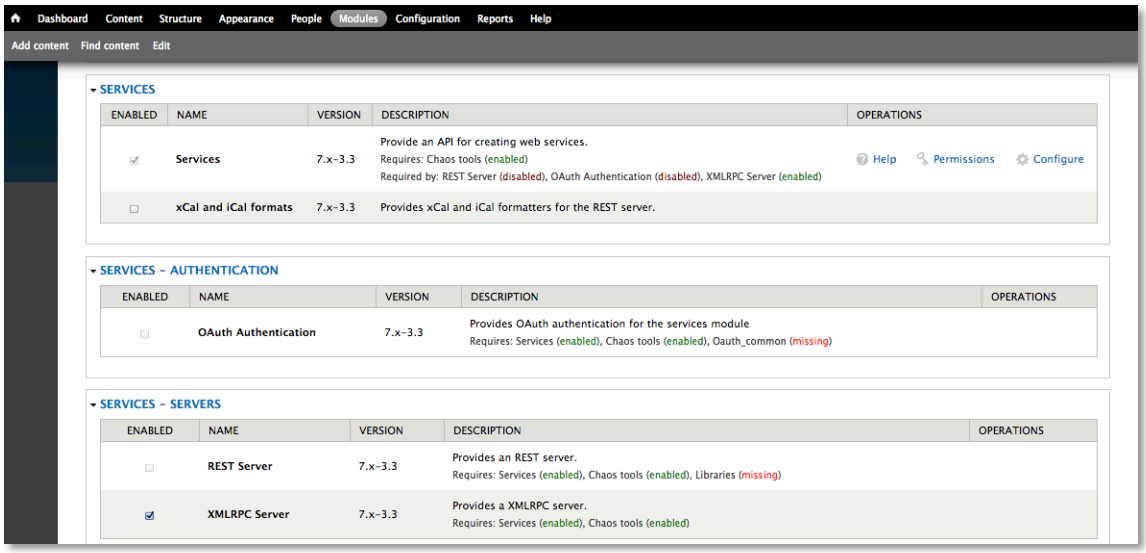


Figura 8.4 Mòduls Services i XMLRPC Server de Drupal

8.2 Connexió amb Drupal

Fent ús dels dos mètodes que inclou el servidor XmlRpc, *system.connect* i *user.login* procedirem a testejar que l'aplicació es connecti correctament amb el servidor.

El mètode system.connect possibilita una connexió inicial amb el servidor, però és només una connexió, és a dir, no autentifica cap usuari sinó que fa l'enllaç amb el servidor com a usuari anònim i ens retorna un identificador de sessió, en forma d'estructura xmlrpc, per indicar que ha funcionat bé. El mètode que sí permet autenticar l'usuari és el user.login, el qual s'ha de cridar després d'haver fet la connexió inicial indicant el nom d'usuari, la contrasenya i l'identificador de sessió que hem obtingut prèviament.

```
[XmlRpcBegin("system.connect")]
public IAsyncResult BeginSystemConnect(AsyncCallback acb)
{
    return this.BeginInvoke(MethodBase.GetCurrentMethod(), new object[] { }, acb, null);
}
[XmlRpcEnd]
public XmlRpcStruct EndSystemConnect(IAsyncResult iasr)
{
    XmlRpcStruct ret = (XmlRpcStruct)this.EndInvoke(iasr);
    return ret;
}

[XmlRpcMethod("user.login")]
public IAsyncResult BeginUserLogin(string Session_id, string username, string password, AsyncCallback acb)
{
    return this.BeginInvoke(MethodBase.GetCurrentMethod(), new object[] { Session_id, username, password }, acb, null);
}
```

Figura 8.5 Fragment de codi on es defineixen els mètodes de connexió

8.2.1 Problemes amb l'ordre dels paràmetres

En provar d'establir connexió i autenticar-se contra el servidor local obtenim un error:

Figura 8.6 Captura de l'error connectant amb el servidor

Tot i haver escrit correctament el usename i el password, que prèviament s'havia donat d'alta a través del web de Drupal, l'aplicació retorna un error d'usuari o contrasenya.

Cal fer èmfasi en que aquesta mena d'errors són els que han consumit gran part del temps dedicat al projecte en ser resolts, provocant així serioses aturades en el desenvolupament del codi.

Gràcies a la instal·lació d'un proxy en l'equip local, que ens permetia capturar el tràfic web d'entrada i sortida a l'equip, es va poder veure on era l'error ja que el servidor retornava informació sobre aquest. El problema era que l'ordre dels paràmetres no era l'adequat. Buscant informació sobre això al web de Drupal es va observar que la versió 6 de Drupal s'espera l'ordre de paràmetres tal i com estaven especificats al fitxer ConnexióDrupalXMLRPC, mentre que la

versió 7, amb la que s'han dut a terme les proves locals, espera un altre ordre: username, password, session_id enlloc de session_id, username, password.

Un cop vist el problema afegim en aquest apartat del codi font un comentari per indicar-ho i poder alternar ràpidament entre servidors només modificant aquesta part del codi.

```
[XmlRpcMethod("user.login")]
public IAsyncResult BeginUserLogin(string Session_id, string username, string password, AsyncCallback acb)
{
    //TODO: l'ordre dels parms a Drupal6 és (session_id, username, password) però a Drupal7 és (username, password, session_id)
    //return this.BeginInvoke(MethodBase.GetCurrentMethod(), new object[] { Session_id, username, password }, acb, null); //EOLYMP
    return this.BeginInvoke(MethodBase.GetCurrentMethod(), new object[] { username, password, Session_id }, acb, null); //LOCAL
}
```

Figura 8.7 Correcció del primer problema en la connexió

Si observem la Figura 8.7 podem veure aquesta correcció feta al fitxer ConnexióDrupalXMLRPC.cs. Afegim al comentari la paraula clau *TODO* per tal que quan li demanem al IDE la llista de tasques que queden per fer (to-do) ens aparegui el comentari.

8.2.2 Problemes amb l'ordre del retorn de user.login

Gràcies al mateix proxy que s'havia fet servir en la resolució de l'anterior problema es va descobrir un altre, també a causa de les diferència de versions entre el Drupal d'eolymp i el de proves local.

La funció user.login retorna una col·lecció valors com a resposta del servidor. Entre aquests valors hi una estructura XML amb la informació personal de l'usuari que ha fet el login, necessària més endavant. El problema trobat en aquesta ocasió era que la versió 6 de Drupal retorna l'estructura amb la informació de l'usuari una posició més endarrerida que la versió 7. En cas de no haver fet res no disposaríem de la informació de l'usuari. Per tant, seguint el mateix funcionament que en el problema anterior, afegim un comentari al codi per notar aquest detall i poder alternar entre versions.

```
[XmlRpcEnd]
public XmlRpcStruct EndUserLogin(IAsyncResult iasr)
{
    XmlRpcStruct ret = (XmlRpcStruct)this.EndInvoke(iasr);
    IEnumerator en = ret.Values.GetEnumerator();
    en.MoveNext(); en.MoveNext();
    //TODO: en Drupal7 cal avançar una posició més ja que el XmlRpcStruct està en segona posició i no en primera
    en.MoveNext();
    ret = (XmlRpcStruct)en.Current;
    return ret;
}
```

Figura 8.8 Correcció del segon problema en la connexió

8.3 Pantalla d'entrenament

Un dels objectius finals d'aquest projectes és afegir una quarta pantalla més, la d'entrenament, a les tres ja existents: competicions, social i perfil.

Aquesta nova pantalla ha de comptar amb el mínim necessari per permetre a l'usuari pujar una activitat al servidor, guardant així el tipus d'activitat, la durada, la data, etc.

Per començar creem la nova vista, de la mateixa manera que es vam crear les tres anteriors (veure capítol 7.2), afegint de moment un únic botó.

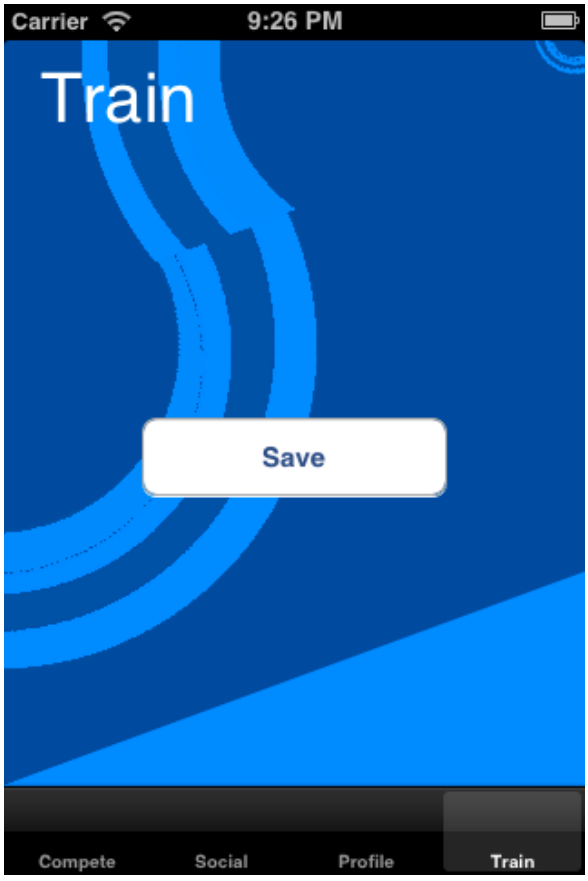


Figura 8.9 Pantalla d'entrenament iPhone

Per la primera prova l'objectiu del botó *Save* serà el de guardar una activitat molt simple, on principalment interessa comprovar que la data de l'entrenament es grava correctament al servidor. El tutor del projecte m'havia comentat que aquest camp havia donat certs problemes amb Drupal 6, i que havíem de provar com es comportava amb la versió 7, la futura versió que tindrà el servidor real d'eolymp.

Per dur a terme aquesta prova al nostre servidor local el primer que cal fer és implementar el concepte d'activitat al Drupal fent servir nodes.

Els nodes són la unitat d'informació bàsica amb que treballa Drupal per poder guardar el contingut, i aquests per defecte contenen la següent informació:

- Autor
- Data de creació
- Títol
- Cos (sumari i cos complet)

D'aquesta manera tot el contingut que es va creant a través de Drupal es guarda en un dipòsit de nodes que anirà creixent conforme es va generant contingut i que proporciona l'avantatge que no cal gestionar-lo, ja que el propi Drupal s'encarrega de tot. Els nodes també poden ser personalitzats per l'administrador i d'aquesta manera, a banda dels camps esmentats abans, podem afegir d'altres segons el nostre interès.

Per tant per tal de poder realitzar la prova necessitarem generar al servidor local un nou tipus de node, que anomenarem **activitat**, i que contindrà un camp més del tipus data, que anomenarem **data_activitat**. La secció del web per afegir nous tipus de nodes es troba a *Administration/Structure/Types* i resulta força intuïtiu.

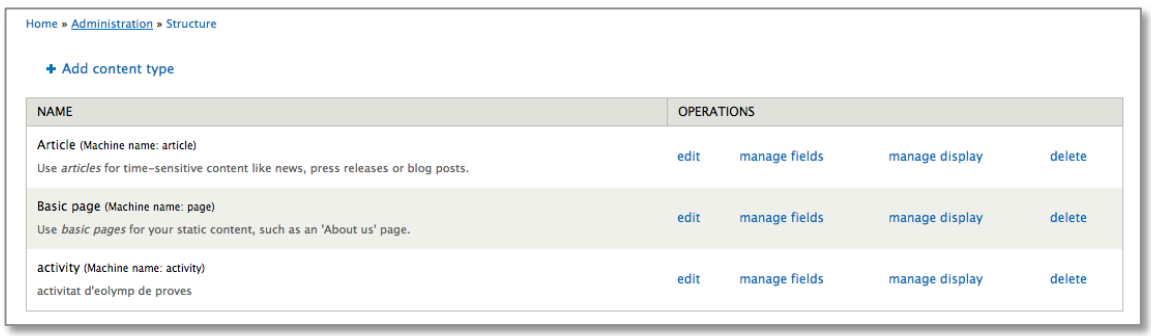


Figura 8.10 Captura de l'apartat de nodes de Drupal

Un cop creat el nou tipus de node, podem afegir els camps necessaris des de l'apartat *manage fields*, que podem veure a la captura anterior, i des d'aquí creem el nou camp de tipus data. Especificarem que el nou camp compti amb les següents propietats:

- Cap requerit a l'hora de guardar el node
- Format d'entrada de la data: 2013-06-21 10:00:00

- Sense cap valor per defecte
- Un únic valor de data per node
- Informació a guardar: any, mes, dia, hores, minuts i segons
- Zona horària: la del servidor (al nostre cas *Europa/Madrid*).
- Forma en que mostrarem el camp guardat: caixa de text

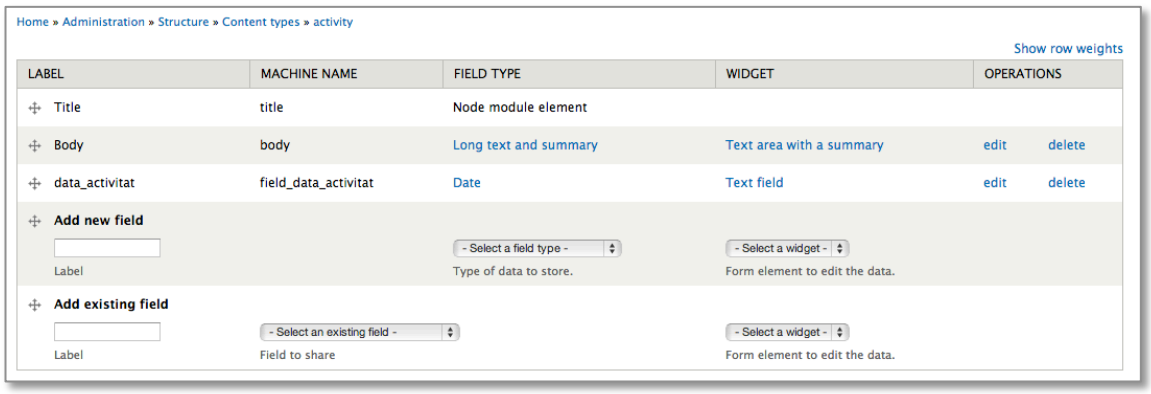


Figura 8.11 Detall del nou camp data_activitat

Ara ja tenim llest el nostre servidor per la prova però encara cal preparar el codi per tal que sigui capaç de gravar un node en aquesta versió de Drupal, ja que el funcionament difereix una mica de la versió anterior. El codi existent ja conté dos mètodes que ens permeten realitzar aquesta tasca:

- **guardarNodeActivitat**: guarda un node del tipus activitat amb els valors que rep com a paràmetre.

- **afegeixCCK**[3]**:** afegeix valor al camp d'un node personalitzat. El camp, el valor i el node al qual s'ha d'afegir es passen per paràmetre.

Seguint l'esquema ja existent crearem dos mètodes nous per tal de poder fer la mateixa funció amb Drupal 7: **guardarNodeDrupal7** i **afegeixCCKDrupal7**. Aquest últim mètode és necessari perquè per guardar un node a Drupal 7 cal espeficar l'idioma en que està la informació, tot dintre del mateix node. A continuació veiem la diferència entre versions a l'hora de guardar un node.

Per la versió 6 de Drupal l'estructura necessària és la següent:

[3] CCK: (Content Construction Kit) permet afegir camps personalitats als nodes

```
XmlRpcStruct value = new XmlRpcStruct();
value.Add ("value",valor);
Object[] objValor={value};
node[key] = objValor;
```

Mentre que per a la versió 7 l'estructura canvia i necessitem afegir el llenguatge:

```
XmlRpcStruct value = new XmlRpcStruct();
value.Add ("value",valor);
Object[] objValor={value};
XmlRpcStruct und=new XmlRpcStruct();
und.Add("und",objValor);
node[key] = und;
```

Aprofitem també per modificar el mètode ja existent *NodeSave*, que fins ara només estava definit per a la versió d'escriptori, afegint el funcionament per a les versions de Windows Phone i d'iPhone, tot fent servir les directrius de compilació condicional.

També cal definir els mètodes que ofereix Drupal 7 per operar amb els nodes dins del nostre codi en forma de proxy (capítol 3.4), ja que aquests no estaven definits i a més a més també tenen diferències entre versions.

```
//metodes node a drupal7: node.retrieve .create .update .delete .index

[XmlRpcMethod("node.create")] //amb drupal6 es node.save
public IAsyncResult BeginNodeSave(XmlRpcStruct node, AsyncCallback acb)
{
    return this.BeginInvoke(MethodBase.GetCurrentMethod(), new object[]{node},acb,null);
}

public XmlRpcStruct EndNodeSave(IAsyncResult iasr)
{
    XmlRpcStruct ret=(XmlRpcStruct)this.EndInvoke(iasr);
    return ret;
}

[XmlRpcMethod("node.retrieve")]
public IAsyncResult BeginNodeRetrieve(string nid,AsyncCallback acb)
{
    return this.BeginInvoke(MethodBase.GetCurrentMethod(),new object[]{nid},acb,null);
}

public XmlRpcStruct EndNodeRetrieve(IAsyncResult iasr)
{
    XmlRpcStruct ret = (XmlRpcStruct)this.EndInvoke(iasr);
    return ret;
}
```

Figura 8.12 Fragment de codi amb els nous mètodes Drupal

Un cop ja tenim tot el necessari per a poder funcionar, programem el botó que havíem afegit a la pantalla d'entrenament perquè al fer clic guardi una data al nou camp data, fent ús dels nous mètodes anteriorment descrits, i executem la prova, obtenint el següent resultat:

Figura 8.13 Captura d'error al guardar la data

El servidor retorna un error indicant que es requereix una data vàlida, i això es perquè la data no està en el format que s'espera. Si desactivem la casella que marca el camp *data_activitat* com a obligatori el node es guarda sense problemes amb la resta d'informació i el servidor no retorna cap error, però la data queda buida.

A continuació veiem una llista amb els diferents valors de les proves que s'han dut a terme, amb idèntic resultat:

- Format de la data:
  - 2013-06-21 10:00:00
  - 2013-06-21T10:00:00
  - 2013-06-21 10:00
  - 2013-06-21T10:00

- Informació a guardar:
  - any, mes, dia, hores, minuts i segons
  - any, mes, dia, hores, minuts

- Zona horària:
  - la del servidor
  - la del camp data
  - la del usuari
  - UTC
  - Sense zona horària

- Forma en que mostrarem el camp guardat:
  - caixa de text
  - calendari
  - llista

- Tipus de la data al node:
  - date
  - datetime

- Tipus del camp data al servidor:
  - Date
  - Date (ISO format)
  - Date (UNIX timestamp)

- Versions de Drupal
  - 7.19
  - 7.22

Així doncs, la tasca de guardar un camp de tipus data amb la versió 7 de Drupal queda avortada per falta de temps i passa a l'apartat de millores o futures ampliacions.

9 Avaluació

En aquest apartat es fa una reflexió sobre les metes aconseguides un cop finalitzat el PFC i s'avalua el rendiment, així com les possibles millores i ampliacions del projecte.

9.1 Metes i objectius assolits

- S'ha aconseguit crear una aplicació per iPhone respectant al màxim possible el disseny de l'aplicació per a Windows Phone.

- A les dues aplicacions mòbils, iPhone i Windows Phone, s'ha aconseguit que el codi específic per a cada una estigui reduït al mínim i bàsicament s'hi implementi només el disseny gràfic dels components.

- Mitjançant la reestructuració ha resultat un codi homogeni, ordenat i ben classificat que permetrà una fàcil addició posterior de més plataformes, com ara Android, i un còmode manteniment de les aplicacions.

- S'ha aconseguit migrar tota la solució inicial de Visual Studio 2010 a la última versió de 2012.

- Ara l'aplicació té la capacitat de connectar amb les dues versions de Drupal i això és de gran utilitat, ja que en aquest moments s'està migrant el servidor d'eolymp de la versió 6 a la 7, per tant el codi ja estarà preparat quan la migració estigui completada.

9.2 Possibles millores

A causa del temps limitat del que es disposa per realitzar el projecte s'ha de prioritzar i hi ha detalls que es van postergant i anul·lant al final que podrien servir per millorar l'aplicació en un futur:

- Es podria millorar el disseny gràfic de les aplicacions, tant de Windows Phone com d'iPhone, i organitzar els continguts d'una manera visualment més atractiva per a l'usuari.

- Extreure més part del codi específic de cada plataforma per tal d'unificar-lo encara més, fins i tot incloent-hi el disseny gràfic comú a totes les plataformes fent servir esquemes XML.

- Afegir a l'aplicació més possibilitats, com ara la de poder pujar un entrenament al servidor, ser capaç de sincronitzar amb un pulsòmetre o xip i ser capaç de guardar les rutes realitzades fent ús del GPS del dispositiu.

9.3 Valoració personal

Aquest projecte m'ha servit per introduir-me en el món del desenvolupament per a mòbils, que a dia d'avui es troba en un punt àlgid.

Des de que els smartphones van fer acte de presència a les nostres vides com a simples telèfons mòbils les seves utilitats han augmentat fins a convertir-se en l'ordinador de butxaca amb infinites possibilitats que son avui dia, i com a programador, trobo que és un món fascinant per endinsar-s'hi.

Abans de començar aquest projecte sabia que volia fer una aplicació per a smartphone però a causa de la diversificació de sistemes operatius que existeix no sabia per quin decantar-me, tenint en compte que és necessari un aprenentatge previ per a qualsevol d'elles, en especial per desenvolupar per a iPhone. Per a mi la troballa del desenvolupament unificat ha estat fascinant a l'hora que un avantatge al no haver de decantar-me només per una plataforma, sinó que aprenent només una manera de fer pots crear una aplicació multi-plataforma.

Tot i les facilitats que brinda la unificació, la feina ha estat laboriosa, i vull destacar que el nombre d'hores recavant informació per aprendre i per resoldre els problemes que han anat sorgint ha estat elevat.

Tot i això, ho recomano a tothom que es vulgui introduir a la programació per a smartphones i vulgui aprendre a desenvolupar aplicacions multi-plataforma.

10 Planificació temporal

En aquest apartat especificarem mitjançant un diagrama de Gantt la planificació temporal del projecte, dividida segons les tasques realitzades.

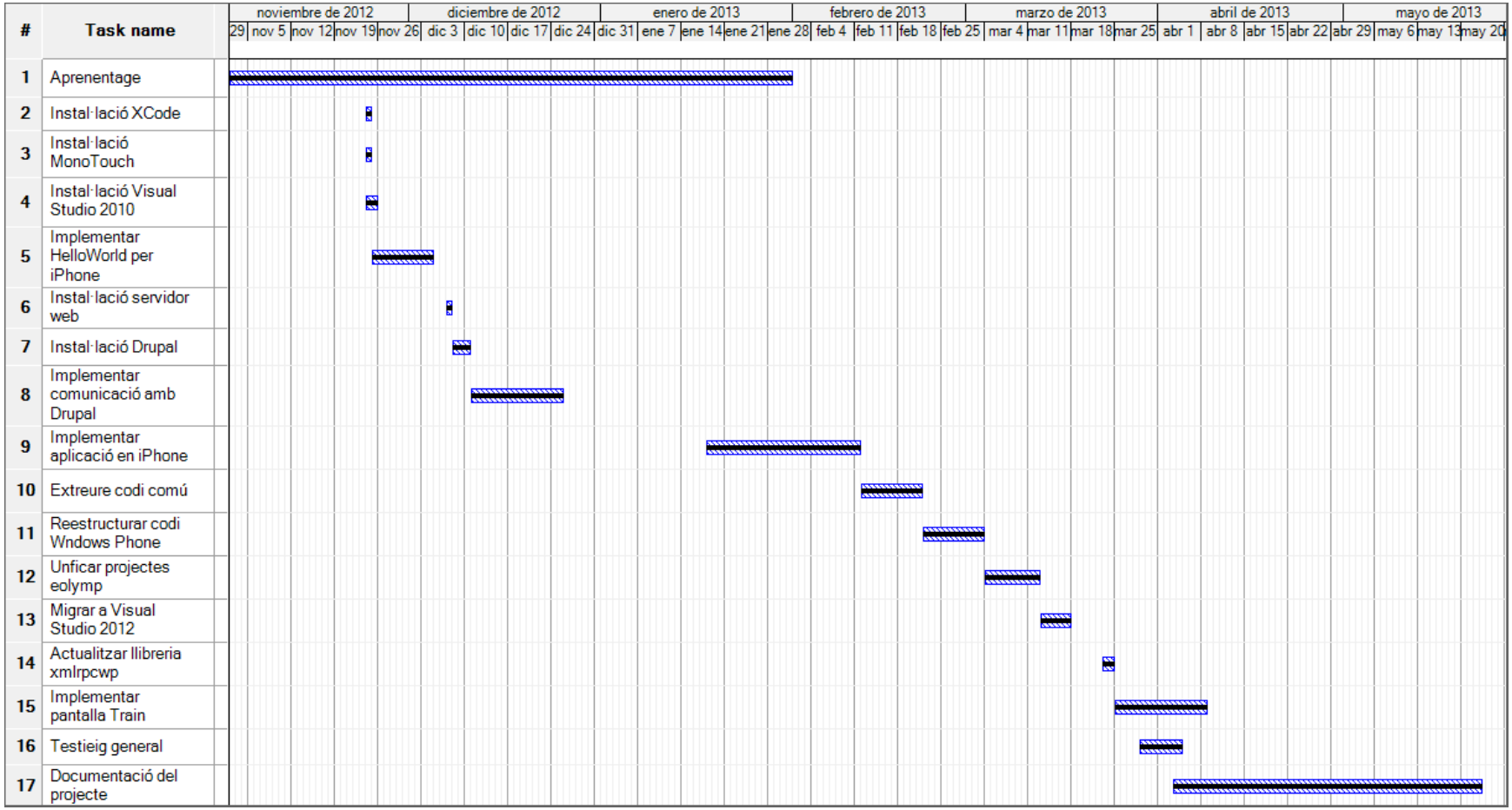


Figura 10.1 Diagrama de Gantt de la planificació temporal

11 Valoració econòmica

En aquest apartat es valorarà econòmicament els recursos emprats que caldrien per haver dut a terme el projecte en un entorn real.

11.1 Materials emprats

Llistat dels materials hardware i software que han estat necessaris per realitzar aquest projecte i el seu cost:

Concepte	Cost
MacBook Pro 13	€ 1226
Parallels Desktop 8	€ 49
Llicència Xamarin.iOS (estudiant)	€ 76
Visual Studio 2012 (estudiant)	Gratuït
Drupal	Gratuït
MAMP	Gratuït
Xcode	Gratuït
TOTAL	€ 1351

Taula 1 Cost material

El cost total del material és una estimació, ja que tant el MacBook Pro com el Parallels per fer rodar Windows en el mateix ordinador són propietat del programador, així que l'únic cost real seria la llicència de Xamarin.iOS.

Cal tenir en compte també que totes les proves s'han dut a terme amb els simuladors, tant de Windows Phone com d'iPhone, i en el moment en que es vulgui poder provar l'aplicació d'iPhone en un dispositiu real o afegir aquesta a la tenda d'aplicacions App Store serà necessària una llicència de desenvolupador d'Apple, amb un cost de 99$ anuals.

11.2 Recursos humans

Totes les tasques realitzades durant aquest projecte es podrien categoritzar segons el tipus de tasca i així poder assignar un perfil a cadascuna per obtenir un preu estimat del que hagués costat la realització real de l'aplicació dins d'un projecte d'enginyeria. Podríem definir els perfils en les següent categories:

- Analista: s'encarrega d'analitzar l'estudi inicial i el disseny general i és el responsable del correcte funcionament de l'aplicació.

- Programador: s'encarrega del desenvolupament del codi de l'aplicació seguint les directrius de l'analista.

- Dissenyador gràfic: s'encarrega de desenvolupar el disseny gràfic proposat per l'analista.

A continuació especifiquem una taula amb una estimació de les hores dedicades segons els perfil de l'enginyer juntament amb una estimació del seu cost per hores:

Perfil	Hores	Cost per hora	Cost
Analista	150	€ 15	€ 2250
Programador	300	€ 10	€ 3000
Dissenyador gràfic	20	€ 10	€ 200
TOTAL	470		€ 5450

Taula 2 Cost humà

11.3 Cost total

Especifiquem a continuació una taula amb el cost total estimat del projecte, tenint en compte els recursos materials i els recursos humans:

Concepte	Cost
Recursos materials	€ 1351
Recursos humans	€ 5450
TOTAL	€ 6801

Taula 3 Cost total

12 Bibliografia

12.1 Llibres

- PROFESSIONAL IPHONE PROGRAMIN WITH MONOTOUCH AND .NET/C#
  Wallace B. McClure, Martin Bowling, Craig Dunn, Chris Hardy, Rory Blyth
  Wiley Publishing, Inc., 2010

12.2 Webs

- jamiebriant / VsMono – GitHub
  https://github.com/jamiebriant/VsMono

- Codigo Monki .NET – Una introducción a MonoTouch
  http://codigomonki.net/una-introduccion-a-monotouch/

- Follesoe / VSMonoTouch – GitHub
  https://github.com/follesoe/VSMonoTouch

- MonoTouch Examples
  http://monotouchexamples.com/

- Chrisntr / Monotouch-Examples – GitHub
  https://github.com/chrisntr/Monotouch-Examples

- MonoTouch in Visual Studio
  http://blog.manniat.net/post/2009/11/18/MonoTouch-in-Visual-Studio.aspx

- Mono hispano
  http://www.mono-hispano.org/

- Xamarin / monotouch-samples – GitHub
  https://github.com/xamarin/monotouch-samples

- Desarollo de software unificado en la era post-pc
  http://www.youtube.com/watch?v=jYp99i4cUDU

- Developing with MonoTouch on Windows and Visual Studio
  http://escoz.com/blog/developing-with-monotouch-on-windows-and-visual-studio

- Xamarin Developer Center
  http://docs.xamarin.com/

- Monotouch tips and snippets
  http://www.yetanotherchris.me/monotouch/monotouch-tips-and-snippets

- Guide to building iOS application with MonoTouch for the .NET/C# developer
  http://devproconnections.com/content1/topic/guide-to-building-ios-applications-with-monotouch-for-the-net-c-developer/catpath/mobile-development/page/2

- Blog de Enrique Aguilar Vargas
  http://www.enriqueaguilarvargas.com/

- Porting a Windows Phone app to iOS
  http://damianblog.com/2012/07/24/porting-a-windows-phone-app-to-ios/

- XML-RPC.NET
  http://xml-rpc.net/

- Drupal documentation
  https://drupal.org/documentation

- Remote operations on Drupal objects
  http://www.codeproject.com/Articles/553121/Remoteplusoperationsplusonplus Drupalplusobjects-pl

- Cocoa XML-RPC Client
  http://www.ditchnet.org/xmlrpc/

- Microsoft msdn library
  http://msdn.microsoft.com/library/default.aspx

- iOS Dev Center
  https://developer.apple.com/devcenter/ios/index.action

- stackoverflow
  http://stackoverflow.com/

- Visual SVN
  http://www.visualsvn.com/visualsvn/

- International Data Corporation
  http://www.idc.com/

- ChampionChip
  http://www.championchip.cat/web/

13 Agraïments

Vull agrair a tota la gent que m'ha donat suport durant tota la carrera, tant familiars com amics, i en especial, als que m'han insistit en que tanqués per fi aquesta etapa de la meva vida amb la presentació d'aquest projecte, sobretot als meus pares. Gràcies.

Dedico un agraïment molt especial a la meva parella, que m'ha donat empenta per realitzar el projecte, ha tingut paciència durant tota la seva durada i fins i tot m'ha ajudat en la revisió d'aquest document. Neus, per a tu.

Per últim vull agrair també el suport del meu tutor durant el projecte, Pau Fonseca i Casas, que ha tingut moltíssima paciència amb mi i m'ha sabut guiar i orientar durant aquesta tasca, alhora que m'ha fet descobrir el desenvolupament multi-plataforma. Gràcies, Pau.

14 Annexes

14.1 Índex de figures

14.2 Índex de taules